Управление проектами по созданию программного обеспечения

Унифицированный подход

УОКЕР РОЙС
RATIONAL SOFTWARE CORPORATION

Издательство "«ЛОРИ»

Содержание

Часть I

Возрождение дисциплины управления разработкой ПО

1 Традиционное управление разработкой ПО 4
1.1 Водопадная модель 5
1.1.1 В теории 5
1.1.2 На практике 11
1.2 Эффективность традиционного управления проектами 17

2 Эволюция экономики разработки ПО 21
2.1 Экономика ПО 22
2.2 Практическая оценка стоимости ПО 26

3 Совершенствование экономики разработки ПО 30
3.1 Уменьшение размера программного продукта 33
3.1.1 Языки 33
3.1.2 Объектно-ориентированные методы и визуальное моделирование 36
3.1.3 Повторное использование 37
3.1.4 Коммерческие компоненты 39
3.2 Совершенствование процесса создания ПО 41
3.3 Увеличение эффективности работы команды 44
3.4 Повышение уровня автоматизации за счет среды разработки . . . 47
3.5 Достижение необходимого качества 50
3.6 Экспертные оценки: взгляд с практической точки зрения 53

4 Два пути: старый и новый 56
4.1 Принципы традиционной программной инженерии 56
4.2 Современные принципы управлениясозданием ПО 65
4.3 Переход к итерационному процессу 69

Часть II

Основы управления процессом создания ПО

5 Стадии жизненного цикла 76
5.1 Стадии разработки и изготовления 77
5.2 Начальная стадия 79
5.3 Стадия уточнения 80

5.4 Стадия конструирования 82
5.5 Стадия ввода в действие 83

6 Рабочие продукты процесса 86
6.1 Комплекты рабочих продуктов 87
6.1.1 Комплект управления 88
6.1.2 Комплекты разработки 89
6.1.3 Эволюция рабочих продуктов в течение жизненного цикла 96
6.1.4 Рабочие продукты, связанные с тестированием 98
6.2 Рабочие продукты управления 100
6.3 Рабочие продукты разработки 108
6.4 Рабочие продукты, используемые на практике 111

7 Архитектура ПО, основанная на моделях 115
7.1 Архитектура с точки зрения управления 116
7.2 Архитектура с технической точки зрения 118

8 Рабочие процессы 123
8.1 Рабочие процессы 124
8.2 Рабочие процессы в рамках отдельной итерации 127

9 Контрольные точки процесса 132
9.2 Второстепенные контрольные точки 140
9.3 Периодические оценки состояния 141

Часть III

Дисциплины управления процессом создания ПО

10 Планирование итерационного процесса 148
10.1 Декомпозиция работ 149
10.1.1 Проблемы традиционной WBS 149
10.1.2 Эволюционирующие декомпозиции работ 152
10.2 Методические рекомендации по планированию 156
10.3 Процесс оценки сроков и затрат 159
10.4 Процесс планирования итераций 160
10.5 Практическое планирование 163

11 Проектные организации и распределение обязанностей . . . 165
11.1 Организации различных отраслей промышленности 166
11.2 Проектные организации 169
11.3 Эволюция организаций 176

12 Автоматизация процесса 178
12.1 Инструменты: «кирпичики» автоматизации 180
12.2 Среда проекта 183
12.2.1 «Круговая» разработка 185
12.2.2 Управление изменениями 186
12.2.3 Различные виды инфраструктуры 193

12.2.4 Среда для других заинтересованных сторон 196

13 Метрики и контроль качества 199
13.1 Семь основных метрик. 200
13.2 Показатели управления 202
13.2.1 Работа и прогресс 203
13.2.2 Предусмотренные в бюджете расходы и затраты. 204
13.2.3 Динамика изменений в командах и штатном расписании . 208
13.3 Показатели качества. 209
13.3.1 Интенсивность изменений и стабильность 209
13.3.2 Дефекты и коэффициент дефектности 209
13.3.3 Доработки и адаптируемость 210
13.3.4 MTBF и завершенность. 211
13.4 Ожидаемое поведение на протяжении жизненного цикла. 212
13.5 Практическое использование метрик ПО 213
13.6 Автоматизация метрик. 215

14 Адаптация процесса 222
14.1 Дискриминанты процесса 222
14.1.1 Масштаб 224
14.1.2 Сотрудничество или соперничество заинтересованных сторон. 227
14.1.3 Гибкость или жесткость процесса. 229
14.1.4 Зрелость процесса. 230
14.1.5 Архитектурный риск. 231
14.1.6 Опыт в предметной области. 232
14.2 Пример: небольшой проект в сравнении с крупномасштабным проектом 233

Часть IV
Взгляд в будущее

15 Особенности современных проектов 240
15.1 Постоянная интеграция 241
15.2 Раннее разрешение рисков 243
15.3 Изменяющиеся требования 244
15.4 Работа различных заинтересованных сторон в одной команде 245
15.5 Десять самых важных принципов управления созданием ПО . . . 247
15.6 Лучшие практические приемы управления созданием ПО 249

16 Экономика ПО следующего поколения 252
16.1 Модели стоимости следующего поколения 253
16.2 Современная экономика ПО. 258

17 Переход к использованию современного процесса 262
17.1 Изменения в общей культуре 263
17.2 Подведение итогов 267

Часть V

Практические примеры и вспомогательные материалы

A Состояние дел в области управления созданием ПО 274

B Модель оценки стоимости COCOMO 280
B.1 COCOMO. 281
B.2 Модель Ada COCOMO 285
B.3 COCOMO II 290

C Метрики изменений 298
C.1 Общий обзор. 299
C.2 Получение метрик 301
C.2.1 Сбор статистики 303
C.2.2 Метрики качества конечного продукта 307
C.2.3 Внутренние показатели прогресса 309
C.3 Использование метрик изменений на практике 313

D Практический пример: CCPDS-R 315
D.1 Общие положения для данного практического примера 316
D.2 Общий обзор подсистем. 317
D.3 Организация проекта 321
D.4 Обзор Подсистемы общего назначения. 322
D.5 Обзор процесса 327
D.5.1 Управление рисками: содержание версии 329
D.5.2 Пошаговый процесс проектирования 332
D.5.3 Эволюция компонентов 335
D.5.4 Процесс пошагового тестирования 338
D.5.5 Рабочие продукты, регламентированные стандартом DOD-STD-2167A Министерства обороны 341
D.6 Оценка, основанная на демонстрации 344
D.7 Главные метрики. 356
D.7.1 Прогресс разработки 357
D.7.2 Прогресс в тестировании 360
D.7.3 Стабильность. 363
D.7.4 Коэффициент дефектности 363
D.7.5 Адаптируемость 364
D.7.6 Завершенность. 365
D.7.7 Затраты финансов/работы на отдельные виды деятельности 366
D.8 Другие метрики 368
D.8.1 Эволюция размера ПО 369
D.8.2 Совершенствование процессов создания подсистем. . . . 373
D.8.3 Диаграмма выполнения SCO. 375
D.8.4 Продуктивность и факторы качества для различных CSCI . 376
D.9 Человеческий фактор 378
D.9.1 Основная команда 378

D.9.2 План постепенного увеличения вознаграждения 379
D.10 Заключение 381

E Усовершенствование процесса и соответствие CMM 385
E.1 Общий обзор CMM 385
E.2 Практическое улучшение процесса 388
E.3 Вопросник для определения зрелости 390
E.4 Вопросы, которые не вошли в «Вопросник для определения зрелости» 411
E.5 Общая оценка процесса 413

Глоссарий 415

Ссылки 422

Предисловие

Эта книга прокладывает путь к следующему поколению методов управления проектами по созданию программного обеспечения (ПО). Многие организации остаются верными модели «водопада», поскольку, несмотря на все свои недостатки, она содержит наиболее полно разработанные руководящие указания о том, как поступать в каждой конкретной ситуации при создании ПО.

До недавнего времени непросто было найти подробное описание альтернативного подхода к управлению, который позволял бы решать такие вопросы, как интеграция компонентов, повторное использование ПО, управление рисками и эволюционные/итерационные/спиральные процессы, при создании ПО. В этой книге предлагается новый проверенный на практике подход и даются указания по его использованию.

Уокер Ройс разработал и опробовал данный подход к управлению созданием ПО в процессе своего участия в большом успешном проекте CCPDS-R, выполненном компанией TRW для военно-воздушных сил Соединенных Штатов. Этот подход был отточен и обобщен при осуществлении самых разнообразных правительственных, аэрокосмических и коммерческих проектов в корпорации Rational.

В главах с 1 по 4 дается обоснование нового подхода; показано, каким образом он позволяет управлять ключевыми для экономики создания ПО моментами по сравнению с традиционным подходом к управлению подобными проектами. Он обеспечивает, во-первых, уменьшение объема ПО, которое требуется создать, во-вторых, снижение количества переделок за счет совершенствования процессов и работы команды и, в-третьих, уменьшение интенсивности труда за счет его автоматизации.

В главах с 5 по 10 представлены особенности новой организации жизненного цикла ПО, которые к тому же служат основой унифицированного процесса создания ПО корпорации Rational (Rational Unified Process).

В нем сочетаются гибкость спиральной модели с дисциплиной управления рисками и набором основных стадий и контрольных точек жизненного цикла ПО, которые касаются основных управленческих решений, принимаемых в процессе создания ПО.

Подобно нашему подходу с анкерными точками (Anchor Point), принятому в University of Southern California (USC), одной из основных контрольных точек жизненного цикла является принятие решения об использовании при разработке архитектуры ПО результатов бизнес-анализа (или о не использовании — в этом случае проект благополучно заканчивается ничем). Контрольная точка, совпадающая с окончанием разработки архитектуры, включает в себя принятие решения о переходе к полномасштабной разработке, которая основывается на создании и описании стабильной архитектуры и на разрешении вопросов относительно всех основных источников риска. Контрольная точка, соответствующая обретению продуктом начальных функциональных возможностей, включает в себя решение о начале бета-тестирования продукта с участием внешних пользователей или эквивалентной ему процедуры.

Автор показывает, чем эти контрольные точки отличаются от точек традиционного подхода, ориентированного на создание документации или на написание кода. Ключевые результаты, материализуемые в ходе создания продукта (требования, проектные решения, реализация, ввод в действие), согласованно изменяются и соединяются в единое целое таким способом, который не противоречит целям проекта и принятой в рамках этого проекта стратегии управления рисками.

В главах с 10 по 14 автор рассказывает, как можно удостовериться в том, что рабочие продукты, касающиеся управления проектом, тоже изменяются согласованно, соединяясь в единое целое. Сюда входят планы проекта и соответствующие оценки затрат финансов и времени, деятельность по организации и созданию команды, параметры проекта, оснащение инструментальными средствами и процессы контроля. Особенно примечательной является глава 14. В ней не только объясняется тот факт, что решения, принимаемые в области управления, зависят от ситуации, но также даются указанию относительно того, каким образом следует учитывать масштаб проекта, культуру команды, зрелость процесса, риск, заложенный в архитектуре, и опыт, имеющийся в данной предметной области.

В главах с 15 по 17 автор пытается понять, в каком направлении движутся лучшие разработчики ПО в своей практике: в сторону управления продуктовой линией, в сторону разработки по «круговому» принципу (round-trip engineering) или в сторону уменьшения размеров команд разработчиков, в которых менеджеры являются одновременно исполнителями, а контроль за качеством становится делом каждого. В приложениях рассматриваются современная практика управления разработкой ПО, семейство моделей для оценки стоимости COCOMO и COCOMO II, а также модель технологической зрелости (Capability Maturity Model, CMM), предложенная Институтом программной инженерии (Software Engineering Institute, SEI). В приложении D дается убедительный практический

пример успешного использования данного подхода при осуществлении широкомасштабного, технически сложного проекта CCPDS-R.

Ройс имеет свежий непредвзятый взгляд на некоторые причуды, глупости и излишества в сфере создания ПО. Это находит свое отражение в «практических» разделах, посвященных таким темам, как оценка стоимости ПО, проверки, рабочие продукты, планирование и метрики. Не все согласятся с мнением автора, однако его оценки являются весьма резкими и способствуют развитию дискуссии.

Мне чрезвычайно повезло в том, что довелось поработать вместе с Уокером Ройсом и с его отцом Уинстоном Ройсом, изучать их опыт и общаться с ними в процессе изменения их путеводных идей.

Барри Боэм
Директор USC Center for Software Engineering
Апрель 1998

Введение

Индустрия ПО неумолимо движется в сторону новых методов управления проектами все возрастающей сложности. В прошлом мы сталкивались с эволюциями, революциями и повторяющимися темами успеха и неудачи. В то время как технологии программирования, процессы и методы быстро развиваются, разработка ПО остается процессом с интенсивным использованием людского труда. Следовательно, способы управления людьми, технологией, ресурсами и рисками обладают большим запасом потенциальных возможностей.

В этой книге управление созданием ПО рассматривается в ракурсе, при котором особое внимание уделяется балансу между следующими элементами:

- Теория и практика
- Технология и люди
- Стоимость для заказчика и прибыльность для производителя
- Стратегия и тактика

В книге читатель будет постоянно встречаться с аспектом управления первостепенной важности: с понятием *баланса*. Необходимо обеспечить баланс между целями заинтересованных сторон, которые общаются между собой с помощью различных языков и систем обозначений. Именно в этом заключается мотивация к искусству открытия изображения камня Розетты. Разработке ПО присущи три фундаментальных языка представления: требования (язык проблемной области), проектные решения (языки трансформации, используемые разработчиками ПО) и реализация (язык, понимаемый компьютерами). Так же, как камень Розетты дал возможность расшифровать египетские иероглифы, методы управления разработкой ПО позволяют перевести возникшую проблему в решение, удовлетворяющее все заинтересованные стороны.

Для управления разработкой ПО не придумано поваренных книг. Не существует рецептов поиска заведомо правильных решений. Я попытался рассмотреть вопросы, вложив в них столько науки, реализма и опыта, сколько было возможно, но управление во многом является предметом рассуждений, (не)здравого смысла и принятия решений в зависимости от ситуации. Вот почему менеджерам платят большие деньги.

Некоторые главы включают в себя разделы с практическим и зачастую очень подробным разбором какой-то конкретной темы. Для того чтобы

отличать эти жизненные руководства от общих моделей процессов, методов и дисциплин, заголовки таких разделов содержат слово «*практический*». Под этим словом я понимаю «лишенный иллюзий» и «честно отражающий действительность», что в точности соответствует цели данных разделов. В них содержатся резкие мнения и провокационные позиции, которые должны задеть за живое читателей, приверженных к каким-либо устаревшим или давно изменившимся решениям, инструментам и технологиям.

Я попытался показать различие между проверенными технологиями, новыми подходами и устаревшими технологиями с помощью доказательств. В большинстве случаев я подкрепляю свою позицию простыми соображениями экономического характера и здравым смыслом, а также различными анекдотами из опыта практического использования. Большая часть материала состоит из уроков, извлеченных из практики управления успешными проектами по созданию ПО в течение последних десяти лет. С другой стороны, некоторый материал представляет новые теоретические положения и гипотетические подходы, которые еще не прошли достаточной проверки практикой.

Я долго колебался, как определить эту книгу — как *учебник* по управлению или как *практическое руководство* по управлению. Различие может показаться ничтожным, но оно очень важно. Проиллюстрирую это примером, который я услышал лет пятнадцать тому назад. Представьте себе, что ваша 14-летняя дочь приходит однажды из школы и спрашивает: «Мам, пап, можно я буду посещать в школе курс сексуального обучения?» Ваша реакция будет совершенно иной, если она спросит: «Можно я буду посещать в школе практические занятия по сексуальному обучению?» (Тогда это значило для меня меньше, чем значит сейчас, когда у меня три дочери-тинэйджера!)

Практическое руководство содержит в себе аспект прикладного знания, что делает эти знания полезными немедленно. Обучение же больше сосредоточено на том, чтобы научить принципам, заложить экспериментальную базу и привить дух предмета, оставив применение этих знаний на потом. Я пытался сделать книгу средством для обучения управлению проектами по разработке ПО. (Я не уверен, что существует такое понятие, как практические занятия по управлению, отличные от опыта, полученного на работе.) Я не претендую на то, чтобы мой совет мог быть непосредственно применим к любому проекту. Я пытаюсь доказать как можно больше утверждений, но некоторые из них так и остались недоказанными, в виде чистых гипотез. Надеюсь, что мои предположения и советы будут способствовать дальнейшим обсуждениям и прогрессу.

Предполагаемая аудитория книги охватывает всех профессионалов-практиков в области ПО. Прежде всего эта книга предназначена для тех, кто принимает решения: тех, кто выделяет инвестиции, и тех, кто расходует бюджеты, выделенные на ПО. В эту группу входят менеджеры-организаторы, менеджеры проекта, лица, ответственные за покупку нового ПО, и их штат. Для этой аудитории я пытаюсь предложить руководства, непосредственно применимые для принятия сегодняшних тактических решений и для определения стратегии завтрашних

инвестиций. Другой важной частью аудитории являются практики, которые принимают и выполняют планы проектов по созданию ПО, а также проводят в жизнь организационные цели и цели проекта.

Стиль

Поскольку книга предназначена для широкой аудитории, я не углубляюсь в технические детали и материалы, многие из которых гораздо лучше обсуждаются в других изданиях. Напротив, я действительно глубоко рассматриваю экономические, управленческие проблемы, стратегию разделения работы на части, стратегию организации и параметры, необходимые для планирования и успешного выполнения проекта по созданию ПО.

Для того чтобы эти сложные темы стали более понятными, в книгу включены иллюстрации. Точность и правильность рисунков требуют некоторых пояснений. В то время как большая часть числовых данных правильно описывает некое понятие, тенденцию, вероятность или связь, формат иллюстраций умышленно сделан неточным. В контексте управления созданием ПО разница между точностью и правильностью не так тривиальна, как это может показаться на первый взгляд, по двум причинам:

1. В процессе управления созданием ПО есть много «серых пятен», т.е. того, что зависит от конкретной ситуации, и двусмысленных решений. Многие понятия трудно, если вообще возможно, правильно изображать, сохраняя точность представления для широкого спектра областей.

2. Понимание различий между точностью и правильностью является фундаментальным качеством хорошего менеджера, управляющего созданием ПО. Он должен предвидеть возможные расходы, риски и эффекты от изменения чего-либо. Неоправданная точность — в потребностях или планах — не раз оказывалась существенным, хотя и трудноуловимым препятствием на пути к успеху.

В моих численных представлениях абсолютные значения не важны и обычно различны в разных областях и обстоятельствах. Основу большинства рисунков и таблиц составляют относительные значения.

При случае я буду приводить в качестве обоснования занимательную историю или реальный практический опыт с тем, чтобы перевести рассмотрение управленческих подходов в практическую плоскость и снабдить их относительно правильными и точными замечаниями об их применимости в зависимости от правил игры. Понять, как технологии, представленные в этой книге, могут быть использованы в реальных условиях, помогут приложения в конце книги. В качестве главного практического примера приводится тщательно документированный удачный широкомасштабный проект. Он показывает, как хорошо могут работать многие из рассматриваемых ниже подходов к управлению. Этот пример содержит также схему для рационализации некоторых усовершенствованных процессов и методов.

Структура книги

Книга состоит из пяти частей, каждая из которых содержит несколько глав:

- **Часть I: Возрождение управления созданием ПО.** В ней описываются современная практика управления созданием ПО и экономика этого процесса, а также вводится понятие переходных состояний, необходимых для гарантированного возврата инвестиций, вложенных в создание ПО.
- **Часть II: Схема процесса управления созданием ПО.** Описываются составляющие процесса и современный подход к управлению созданием ПО, включающий в себя стадии жизненного цикла, результаты, рабочие процессы и контрольные точки.
- **Часть III: Дисциплины управления созданием ПО.** Обобщаются некоторые критичные методы, касающиеся планирования, контроля и автоматизации современного процесса создания ПО.
- **Часть IV: Взгляд в будущее.** Высказываются предположения относительно ожидаемой эффективности выполнения современных проектов и об экономике создания ПО следующего поколения, а также обсуждаются изменения в культуре, необходимые для достижения успеха.
- **Часть V: Практические примеры и вспомогательный материал.** В пяти приложениях даются базовые обоснования некоторых рекомендаций, указаний и мнений, встречающихся в книге.

Благодарности

На мой итерационный процесс разработки оказали влияние многие источники, однако при написании этой книги я обращался к относительно малому числу опубликованных работ. Приводя более обширный список имеющих отношение к данному вопросу публикаций, я надеюсь, что это поможет некоторым читателям и принесет чувство удовлетворения некоторым авторам, хотя большинство совпадений с моими взглядами оказывается чисто случайным.

В основе материала книги лежат три источника, к которым я обращался постоянно:

1. TRW's *Ada Process Model Guidebook* [Royce, Walker, 1989] (Руководство по модели процесса Ada компании TRW). В этой книге я описал успешный процесс реализации большого проекта компанией TRW с тем, чтобы он мог использоваться в компании.
2. Семинар корпорации Rational Sofware по управлению созданием ПО [Royce, Walker, 1997]. Я подготовил материалы для этого семинара, посвященного лучшим примерам создания ПО и рассчитанного на два дня, для того, чтобы описать подход корпорации Rational Software к управлению созданием ПО. Рецензентами были такие

достойные люди, как Дон Андрес (TRW), Барри Боэм (Университет Южной Калифорнии), Ларри Дрюффель (SEI), Ллойд Моузманн (военно-воздушные силы США) и Уинстон Ройс (TRW), не считая практических работников и исполнителей внутри корпорации Rational Sofware. Этот семинар проводился десятки раз в середине 90-х гг. в самых различных аудиториях, включая правительственные группы, военных, ответственных за заключение контрактов, и коммерческие структуры.

3. Технология Rational Unified Process. Приобретение корпорацией Rational технологии Objectory привело к большим внутренним инвестициям в слияние методов Objectory (в основе которых лежит процесс, управляемый вариантами использования (use-case-driven)) и существующего процесса корпорации Rational (охватывающего методы управления и объектно-ориентированное моделирование). Эти инвестиции продолжаются и по настоящее время, поскольку корпорация Rational распространяет описание процесса и рекомендации на все большее число видов деятельности в рамках жизненного цикла, инструментов и методов, что и привело к созданию Rational Unified Process.

Еще несколько источников оказали заметное влияние на описание процесса управления, представленного в данной книге. Это влияние стало результатом долгосрочных взаимоотношений, которые включают в себя многие годы совместной работы, обмена идеями и обширных непосредственных контактов.

- Мой союз с Барри Боэмом в течение последних 15 лет являлся неиссякаемым источником знаний в области программной инженерии.
- Выдающееся лидерство Дона Андреса и его знания и опыт в области управления проектами выделяют его среди многих других менеджеров проектов, с которыми и для которых мне приходилось работать; я узнал от него много нового.
- Дэйв Бернстейн, Роберт Бонд, Майк Девлин, Кевин Хаар, Поль Леви, Джон Ловитт и Джо Мараско – группа высших менеджеров корпорации Rational. Все они обладают живым умом и ясным видением программной инженерии как бизнеса.
- Работа Филиппа Кручтена в области архитектуры ПО и основ процесса, а также его практический опыт способствовали формированию моего понимания и моих представлений.
- Гради Буч, Ивар Якобсон и Джим Рамбо – ведущие методологи корпорации Rational – оказали великую услугу сообществу разработчиков ПО, создав унифицированный язык моделирования UML (Unified Modeling Language).
- Сотни профессионалов корпорации Rational отвечали за практические результаты проектов по разработке ПО и за переход от теории разработки ПО к практике.

Наиболее значимое влияние на эту работу оказал мой отец, Уинстон Ройс, благодаря которому сформировались мои воззрения и утвердились позиции. Он критиковал мои представления, тем самым укрепил мою решимость занять провокационную позицию и способствовал прогрессу.

Еще несколько человек потратили свое время, просматривая ранние версии моей рукописи, и внесли свой вклад в понятия, представления и качества, содержащиеся в этой книге. Моя особая благодарность Али Али, Дону Андресу, Питеру Бишу, Барри Боэму, Гради Бучу, Дугу Ишигаки, Ивару Якобсону, Каперсу Джонсу, Хартмуту Кохеру, Филиппу Кручтену, Эрику Ларсену, Джо Мараско, Ллойду Моузманну, Роджеру Обергу, Ричу Рейтману, Джиму Рамбо и Джону Смиту.

Наконец, общее качество представления, последовательность и понятное изложение материала книги — это все плоды труда Карен Эйлор. Ее критика, организаторские таланты, внимание к частностям и агрессивное выискивание ошибок оказали существенное влияние на окончательное содержание книги.

Часть I

Возрождение дисциплины управления разработкой ПО

Глава 1 ◆ **Традиционное управление разработкой ПО**

Глава 2 ◆ **Эволюция экономики разработки ПО**

Глава 3 ◆ **Совершенствование экономики разработки ПО**

Глава 4 ◆ **Два пути: старый и новый**

Индустрия ПО переживает период возрождения. Многие укоренившиеся принципы, лежащие в основе разработки ПО, вытесняются новыми технологиями или заменяются более совершенными методами или более высоким уровнем автоматизации.

В любой области знаний важно, чтобы изучающий ту или иную дисциплину осознал ее текущее состояние, прежде чем переходить к чему-то новому. Перед изучением подхода к управлению разработкой ПО, который будет применяться в будущем, необходимо понять, каково сегодняшнее состояние индустрии ПО и какими путями она пришла к этому.

За последние 10 лет я принимал участие в попытках усовершенствовать процесс создания ПО в нескольких компаниях, входящих в список Fortune 500. Типичными задачами было достижение 2-, 3- или 10-кратного увеличения производительности, качества, ускорения выхода на рынок (по отдельности или всего сразу). Самое смешное заключается в том, что многие из этих компаний не имели ни малейшего представления об объективном значении своих текущих показателей.

Главы части I дают представление о практическом состоянии индустрии ПО и определяют существующие показатели эффективности для традиционного процесса управления разработкой ПО.

Глава 1

Традиционное управление разработкой ПО

Ключевые моменты

- Традиционный подход к управлению созданием ПО кажется хорошим только в теории; на практике же он основывается на устаревших технологии и методах.
- Традиционная экономика ПО позволяет получить точку отсчета для традиционных принципов управления созданием ПО.

Самым лучшим качеством ПО является его гибкость: запрограммировать можно практически все что угодно. Худшим качеством ПО также является его гибкость. Это самое «практически все что угодно» сильно затрудняет планирование, мониторинг и управление разработкой ПО. Непредсказуемость – первопричина того, что в последние 30 лет называли «кризисом ПО».

В середине 90-х гг. были предприняты по крайней мере три важные попытки анализа состояния индустрии разработки ПО. Результаты были представлены в *Patterns of Software Systems Failure and Success* (Образцы успехов и неудач систем ПО) [Jones, 1996], в «Chaos» («Хаос») [Standish Group, 1995] и в *Report of the Defense Science Board Task Force on Acquiring Defense Software Commercially* (доклад научного совета по обороне, касающийся задачи увеличения количества ПО для оборонных целей, получаемого на коммерческой основе) [Defense Science Board, 1994]. В приложении А приведены некоторые результаты, имеющие отношение к этому вопросу.

Эти три аналитических исследования привели к одному и тому же выводу: процент успешных проектов по созданию ПО чрезвычайно низок. По некоторым моментам имелись разногласия, но основные посылки хорошо согласовывались и дополняли друг друга. Выводы заключались в следующем:

1. Разработка ПО по-прежнему абсолютно непредсказуема. Только около 10% проектов по созданию ПО оказываются успешными, укладываясь в первоначальные бюджетные и временные рамки.
2. Управление определяет успех или неудачу в большей степени, чем технологические преимущества.
3. Количество «выброшенного на свалку» и переделанного ПО является показателем незрелости процесса.

Эти три работы ясно продемонстрировали сложность проблемы и современные нормативные показатели эффективности традиционного управления разработкой ПО. Здесь есть что совершенствовать.

В завершение главы дается обобщение процесса управления созданием ПО, который использовался в большинстве традиционных проектов. Поскольку у этого подхода, известного как водопадная (waterfall) модель, существует много производных, он является основополагающим для большинства проектов, опыт по которым накоплен к настоящему моменту. Обобщать всегда опасно, поэтому методы усовершенствования процесса, обсуждаемые на протяжении книги, должны рассматриваться в подходящем контексте.

1.1 ВОДОПАДНАЯ МОДЕЛЬ

Большинство трудов по программной инженерии представляет водопадную модель как основу «традиционного» процесса создания ПО. Я же склонен рассматривать ее как показатель уровня этого процесса. Сначала мы изучим и подвергнем критике теорию водопадной модели, а затем посмотрим, как традиционный процесс создания ПО используется в промышленных целях. В действительности при промышленном подходе большинство теорий игнорируется, однако водопадной модели удается служить основой для многих удачных примеров (и многих неудачных), особенно при совместном применении с современными технологиями.

1.1.1 В теории

В 1970 г. мой отец, Уинстон Ройс, опубликовал в IEEE WESCON статью «Managing the Development of Large Scale Software Systems» («Управление разработкой широкомасштабных систем ПО») [Royce, Winston, 1970]. Эта статья, основанная на уроках, извлеченных им в процессе руководства большими проектами по разработке ПО, и по сей день остается наиболее цитируемым источником по водопадной модели. В статье представлено глубокое и краткое изложение философии традиционного управления разработкой ПО того времени, и многие из рекомендаций тридцатилетней давности выдержали проверку временем, несмотря на большие изменения в технологии.

В статье были приведены три основополагающих момента. (Цитаты и перефразированные утверждения выделены *курсивом*.)

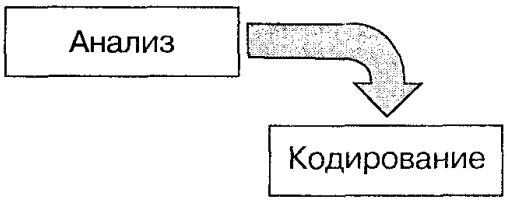
1. *Разработка компьютерных программ состоит из двух основных этапов: анализ и кодирование.*
2. *Для того чтобы иметь возможность управлять и контролировать всю ту интеллектуальную свободу, которая присуща процессу разработки ПО, необходимо ввести несколько «сверхнормативных» этапов, включающих в себя определение системных требований, определение требований к программному обеспечению, разработку программы и тестирование. Эти этапы дополняют этапы анализа и кодирования. На рис. 1.1 показаны структура проекта и основные этапы разработки крупномасштабной программы.*
3. *Основной подход, описываемый водопадной моделью, является весьма рискованным и допускает неудачное завершение. Стадия тестирования, находящаяся в конце цикла разработки, – первый момент, где можно определить реальное время выполнения, объем занимаемой памяти, скорость ввода/вывода и т.д., чтобы сравнить их со значениями, установленными при анализе. Изменения, внесенные в программу, могут оказаться настолько разрушительными, что требования к ПО, на которых основывалась разработка программы, окажутся невыполненными. В таком случае придется либо пересмотреть требования, либо внести существенные изменения в структуру программы.*

Пункт 1, кажущийся по началу тривиальным, ниже будет развернут в одну из моих основных тем, касающихся управления: отделение стадии разработки от стадии производства.

Семь из девяти страниц статьи посвящены описанию пяти условий, которые необходимы для усовершенствования основного водопадного процесса и могли бы исключить многие риски при разработке, указанные в пункте 3. Обсудим подробнее эти пять пунктов. (Цитаты и перефразированные утверждения выделены *курсивом*, за ними следуют комментарии автора в контексте сегодняшней технологии и терминологии.)

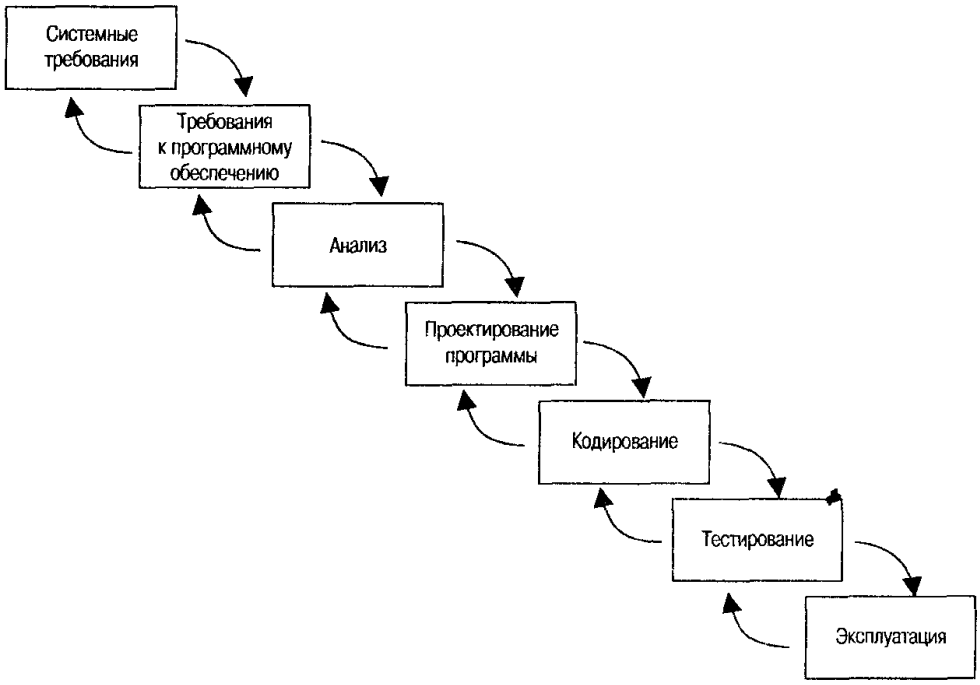
1. ***Проектные решения первичны.*** *Прежде всего необходимо включить стадию предварительного проектирования программы между стадией формирования требований к ПО и стадией анализа. Таким способом разработчик программы может убедиться в том, что она не окажется непригодной из-за изменения объема занимаемой памяти, времени выполнения и данных. При переходе к стадии анализа разработчик программы должен передать аналитику налагаемые ограничения по времени, памяти и операциям таким способом, чтобы тот смог оценить последствия. Если суммарные требуемые ресурсы окажутся неприемлемыми либо если в проект изначально вкрались ошибки, это будет установлено на ранней стадии, и итерация с определением требований и предварительным проектированием может быть повторена до начала основного проектирования, кодирования и тестирования. Каким же образом реализуется эта процедура разработки программы? Необходимо предпринять следующее:*

Водопадная модель, часть 1:
Два основных этапа построения программы



И анализ, и кодирование включают в себя творческую работу, что непосредственно влияет на качество конечного продукта.

Водопадная модель, часть 2: Крупномасштабные системы



Водопадная модель, часть 3:
Пять условий, необходимых для работы этого подхода

1. Завершайте проектирование программы до начала анализа и кодирования.
2. Ведите документацию полно и своевременно.
3. Выполняйте работу дважды, если это возможно.
4. Планируйте, контролируйте и наблюдайте за тестированием.
5. Привлекайте к работе заказчика.

Рис. 1.1. *Водопадная модель*

Начинайте процесс проектирования с разработчиками программ, а не с аналитиками или программистами.

Продумайте, опишите и распределите режимы обработки данных, обращая внимание на риск появления возможных ошибок. Распределите функции обработки, создайте базу данных, установите время

выполнения, определите интерфейсы и режимы взаимодействия с операционной системой, опишите режимы ввода/вывода и предварительно определите эксплуатационные процедуры.

Составьте обзорный документ – понятный, информативный и соответствующий текущему положению вещей – с тем, чтобы каждый работающий над проектом мог получить из него детальное представление о системе.

▲ Существо схемы процесса, представленной в последующих главах, — упреждающая разработка архитектуры (architecture-first development). Хотя некоторые термины могут меняться (например, вместо термина «проект программы» используется термин «архитектура»), существо современного процесса соответствует приведенным здесь объяснениям. Как будет описано ниже, архитектура первична, она определяется и разрабатывается параллельно с планированием и формированием требований, являясь одной из составляющих частей стадии разработки проекта.

2. ***Документируйте проект.*** *Объем документации, необходимой для компьютерных программ, довольно велик; он намного больше того объема, который готовы создать программисты, аналитики или разработчики программ, если оставить их наедине со своими компьютерами. Зачем же требуется так много документации? (1) Каждый разработчик должен обмениваться информацией с разработчиками интерфейсов, менеджерами и, возможно, заказчиками. (2) На ранних стадиях документация – это и есть проект. (3) Истинная ценность документации заключается в том, что она позволяет поддерживать на более поздних этапах изменения, выполняемые отдельной группой тестирования, отдельной группой сопровождения и группой эксплуатации, которые не знают ПО.*

▲ Если не обращать внимания на технологические несоответствия того времени, когда была написана статья, то существо посылки «документируйте проект» остается в силе. Понятное изложение основных моментов, доступное всем исполнителям и группам, по-прежнему остается существенным. Однако по причине больших изменений, произошедших с системами обозначений, языками, браузерами, инструментами и методами, необходимость во многих документах отпала. В последующих главах я отстаиваю ту точку зрения, что сосредоточивать свое внимание на документации ошибочно и непродуктивно. Это является следствием того, что сегодняшние технологии поддерживают скрупулезные и самодокументируемые системы обозначений для описания требований, структур и реализаций.

3. ***Выполняйте работу дважды.*** *Если компьютерная программа разрабатывается впервые, добейтесь того, чтобы версия, которая в конце концов попадет к заказчику для реального использования, была бы на самом деле второй, хотя бы для наиболее критичных проектных/эксплуатационных решений. Заметьте, что это всего лишь миниатюрная копия полного процесса, требующая соответственно малых затрат времени.*

Работающая над первой версией команда должна быть особенно компетентна, что позволит быстро обнаружить проблемные точки в проекте, смоделировать их и альтернативные им решения, не обращая внимания на очевидные аспекты разработки, которые несущественны на ранних стадиях, и в результате получить программу, свободную от ошибок.

▲ Это — краткое и упрощенное описание упреждающей разработки архитектуры. Ответственность за начальную разработку возлагается на «архитектурную группу». Обобщение такой практики приводит к появлению подхода «выполняйте работу N раз», на котором основывается современная итерационная разработка (см. ниже).

Без этого первого прохода менеджер проекта вынужден полагаться на мнение разработчиков. С «моделированием» же первого прохода у него появляется возможность экспериментальной проверки некоторых ключевых гипотез. Но во всем остальном ему придется полагаться на человеческую оценку, которая в области разработки компьютерных программ (так же, как и при определении максимально допустимого взлетного веса, стоимости проекта или победителя на скачках) неизменно и существенно оптимистична.

▲ Это замечательное описание духа итерационной разработки и присущих ей преимуществ в управлении рисками.

4. ***Планируйте, контролируйте и следите за тестированием.*** *Без всякого сомнения, самым главным потребителем ресурсов, выделенных на проект (человеко-дней, машинного времени и/или управленческого анализа), является стадия тестирования. Эта стадия наиболее рискованна с точки зрения стоимости выполнения и соблюдения графика работ. Она обычно является завершающим этапом, когда вернуться назад и пойти по другому пути вряд ли возможно (или вообще невозможно). Три предыдущие рекомендации имели целью поиск и исправление ошибок до начала стадии тестирования. Однако даже при соблюдении этих рекомендаций все равно приходится переходить к стадии тестирования и все равно остаются некоторые важные вещи, которые необходимо выполнить, а именно: (1) задействовать группу специалистов для проведения тестирования, которые не несут никакой ответственности за исходную разработку; (2) выполнить визуальные проверки для обнаружения очевидных ошибок, таких как отсутствие знаков «минус», пропуск множителей двойки, переход по неверному адресу (не следует использовать компьютер для выявления подобного рода ошибок, это слишком накладно); (3) протестировать все логические ветви программы; (4) произвести окончательную проверку на том компьютере, на котором будет применяться программа.*

▲ Среди этих советов есть несколько хороших и несколько устаревших. Пункты 1 и 4 по-прежнему важны. Пункт 2 на сегодняшний день является распространенным способом проверки качества (применение инспекций ПО), но его цель в том виде, в котором она изложена здесь, устарела.

Подобный подход мог быть хорошим и эффективным по стоимости при использовании технологии 1970 г., но не современных подходов. Компьютеры, компиляторы, анализаторы и прочие инструменты являются более эффективными механизмами поиска очевидных ошибок. Что касается пункта 3, то тестирование всех логических ветвей программы и в 1970 г. было уже довольно непростой задачей даже без учета тех сложностей, которые привнесли распределенные повторно используемые компоненты и другие усложняющие факторы. И конечно же, это абсолютно неосуществимо для большинства современных систем. Особенно это касается распределенных вычислений, для которых время является дополнительным измерением, а число логических ветвей практически бесконечно. В современном процессе тестирование происходит на протяжении всего жизненного цикла, что, при правильном его выполнении, требует намного меньше суммарных ресурсов и позволяет находить ошибки на более ранних стадиях жизненного цикла, когда еще существует возможность использования альтернативных решений.

5. ***Привлекайте к работе заказчика.*** *По ряду причин проектные решения являются предметом, допускающим широкую интерпретацию даже после достижения предварительного соглашения. Важно привлечь клиента к работе формальным путем с тем, чтобы он был вовлечен в разработку на ранних ее стадиях и до окончательной сдачи проекта. Существуют три стадии, следующие за определением требований, на которых введение в курс дела, оценки и участие заказчика могут способствовать успеху разработки. Это «предварительный обзор ПО», следующий за этапом предварительной разработки программы, серия «критических обзоров структуры ПО» в процессе разработки программы и «окончательный обзор принимаемого ПО», который идет за тестированием.*

   ▲ Введение в курс дела использовалось в течение многих лет, и везде результаты оказывались положительными. Привлечение заказчика к предварительным демонстрациям и к оценке альфа/бета-версий является проверенной и ценной практикой.

Я всегда восхищался тем, насколько проницательной оказалась эта статья. В то время как многие тратили всю свою энергию, пытаясь опровергнуть подход, предлагаемый водопадной моделью, мне удалось найти минимальное количество изъянов в этой теории, даже с учетом применения ее в контексте современной технологии. Критике следовало бы подвергнуть практику использования этого подхода, которая включает в себя некоторые необоснованные и неработающие элементы. Я подозреваю, что большинство критиков никогда по-настоящему не понимали этой теории; они знакомы лишь с обычной практикой.

На протяжении всей книги я буду ссылаться на практику применения в прошлом и в настоящем того подхода, который предлагается водопадной моделью и который далее при обсуждении называется «традиционным» подходом к управлению процессом создания ПО. Я готов поспорить, что этот процесс не является хорошим подходом для современных разработок и технологий, и я буду использовать его в качестве отправной точки при рационализации процесса с целью его усовершенствования, что позволит избавиться от некоторых его врожденных пороков.

1.1.2 На практике

Несмотря на рекомендации многих экспертов и теорий, более поздних по отношению к водопадной модели, в целом ряде проектов по-прежнему используется традиционный подход к управлению созданием ПО. Поскольку в настоящее время его применение сокращается, я буду говорить о нем в прошедшем времени.

Сформулируем краткие характеристики традиционного процесса в том виде, в котором он обычно использовался, что, вообще говоря, может и не совпадать с тем, как он должен был использоваться на самом деле. Проекты, в которых приходилось сталкиваться с трудностями, зачастую имели следующие проблемы:

- Затянувшаяся интеграция и позднее обнаружение ошибок, допущенных при разработке
- Позднее разрешение рисков
- Функциональная декомпозиция, определяемая требованиями
- Противостояние между участниками проекта
- Чрезмерное внимание, уделяемое документации и совещаниям для обмена мнениями

Затянувшаяся интеграция и позднее обнаружение ошибок, допущенных при разработке

На рис. 1.2 показан ход разработки в зависимости от времени в случае традиционного проекта, использующего водопадную модель управления. Прогресс определяется как процент написанного кода, т.е. того, что можно доказательно признать готовым. (Написанный код может быть транслирован и выполнен, при этом он необязательно является полным, совместимым или соответствующим спецификациям.) В таких случаях обычным является следующее:

- Раннее завершение этапа «бумажного» проектирования и подробная (зачастую излишне) постановка задач
- Переход к кодированию на поздних этапах жизненного цикла
- Кошмарные трудности при интеграции, возникающие из-за непредвиденных проблем при реализации и из-за неточностей в интерфейсах
- Жестко регламентированные бюджет и время сдачи системы
- «Втискивание» неоптимальных решений в тот момент, когда уже не остается времени на перепроектирование
- Трудно сопровождаемый сырой продукт, передаваемый заказчику с опозданием

При тех несовершенных языках и технологиях, которые использовались в традиционном подходе, особое внимание уделялось прежде

Формат	Специальный текст	Блок-схемы	Исходный код	Базовые конфигурации
Деятельность	Анализ требований	Разработка программы	Кодирование и тестирование модулей	Затянувшаяся интеграция и тестирование
Продукт	Документация	Документация	Кодированные модули	Временные настройки

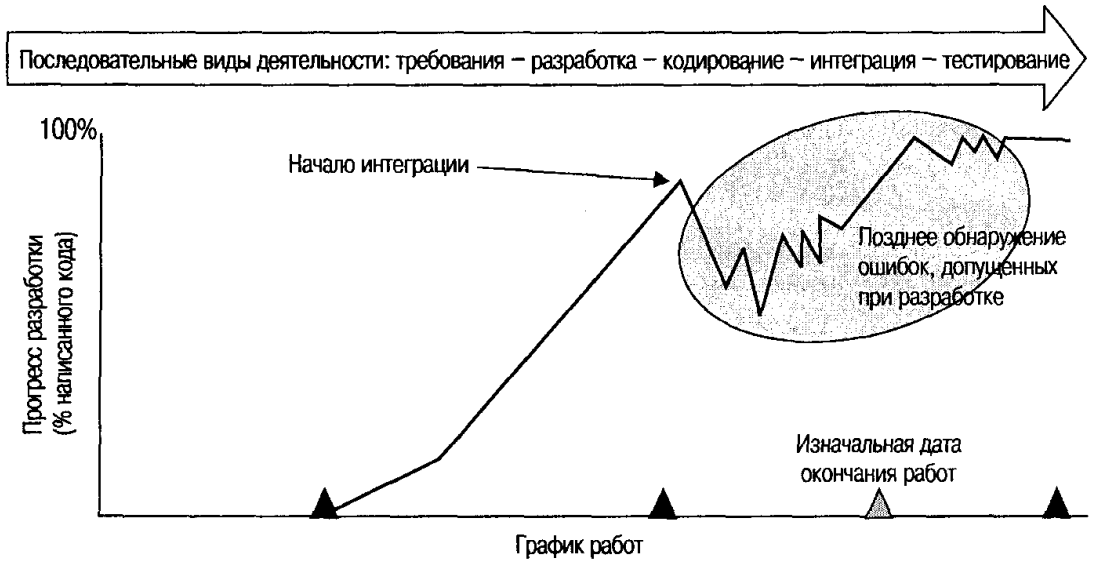


Рис. 1.2. *Ход разработки в традиционном проекте*

всего тщательной «разработке ПО», и только после этого начиналось создание ПО на языке программирования, когда понимание и внесение изменений становится затруднительным. Подобная практика предполагала применение множества различных форматов (требования – на естественном языке, предварительная разработка – в виде блок-схем, детальная разработка – на языках разработки программ, а окончательная реализация – на языках программирования, таких как FORTRAN, COBOL и C) и чреватых ошибками, трудоемких переводов из одного формата в другой.

Традиционные методы, применявшиеся в процессе разработки при использовании водопадной модели, неминуемо приводили к запоздалой интеграции и к устойчивым системным ошибкам в работе программы. В традиционной модели вся система создавалась сначала на бумаге, затем целиком реализовывалась, затем интегрировалась. И только по окончании этого процесса становилось возможным тестирование системы, позволяющее убедиться в том, что основная архитектура (интерфейсы и структура) разработана правильно. Одной из распространенных особенностей проектов, для реализации которых применяется традиционный подход, является то, что на проведение тестирования тратится 40% или более от всех ресурсов. В таблице 1.1 представлено типичное распределение затрат по всему спектру действий, выполняемых при создании ПО.

Таблица 1.1.
Затраты на различные виды деятельности в традиционном проекте

Деятельность	Затраты
Менеджмент	5%
Определение требований	5%
Проектирование	10%
Кодирование и тестирование модулей	30%
Интеграция и тестирование	40%
Ввод в действие	5%
Создание среды разработки	5%
Всего	100%

Позднее разрешение рисков

Одной из серьезных проблем, связанных с водопадной моделью, являлась невозможность раннего разрешения рисков. Это было не столько результатом жизненного цикла водопадной модели в целом, сколько результатом повышенного внимания на ранних стадиях к «бумажным» продуктам, в которых реальные риски разработки, реализации и интеграции оставались относительно малыми. На рис. 1.3 представлена типичная кривая распределения рисков для проектов, использующих водопадную модель. Она разбивается на четыре отдельных периода подверженности рискам, для каждого из которых риск определяется как вероятность

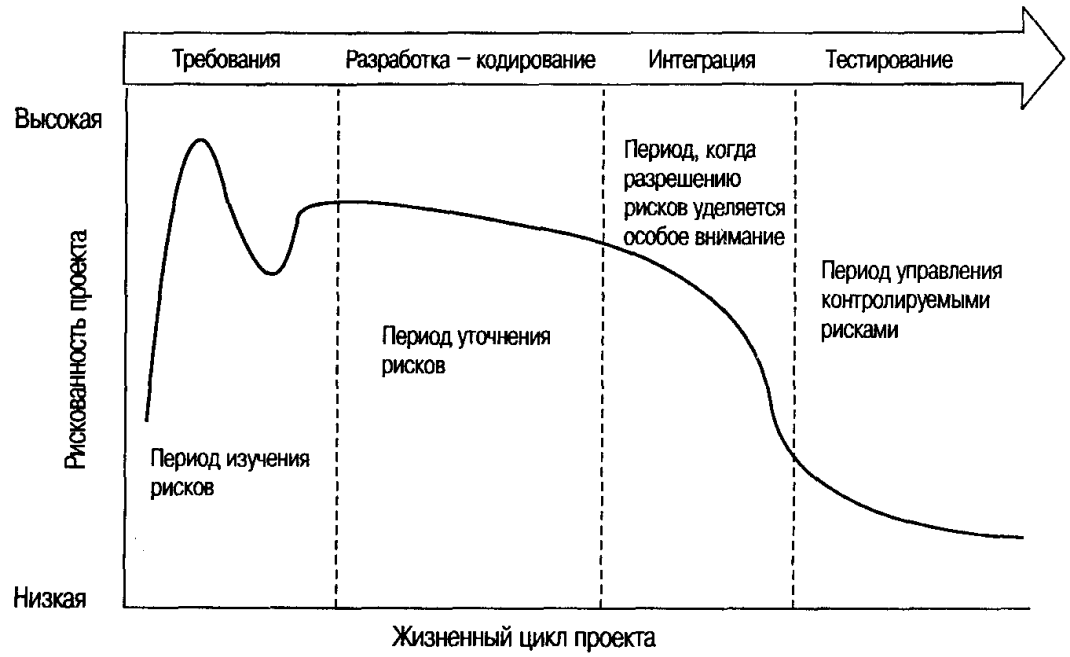


Рис. 1.3. *График риска в традиционном проекте в течение его жизненного цикла*

2*

невыполнения поставленной задачи по смете, срокам, функциональным возможностям или качеству. В самом начале жизненного цикла по мере определения требований реальный уровень риска практически непредсказуем. После того как понимание требований концепции разработки становится устоявшимся, даже если эта концепция существует только на бумаге, уровень риска стабилизируется. Однако стабилизируется он обычно на относительно высоком уровне, поскольку на этот момент времени существует весьма незначительное количество реальных фактов, которые могли бы позволить менеджеру получить объективную оценку. По мере написания кода некоторые риски для отдельных компонентов оказываются разрешенными. После этого осуществляется интеграция, и основную роль начинают играть качество и риски, присущие системе в целом. Обычно именно в течение этого периода удается решить многие проблемы и достичь большинства компромиссов. Однако решение проблем на столь позднем этапе жизненного цикла, когда существует огромная инерция, тормозящая внесение изменений в продукт, оказывается чрезвычайно накладным. Соответственно, проекты имеют тенденцию к затягиванию стадии интеграции (см. рис. 1.2) до тех пор, пока не будут реализованы основные изменения в проекте (перепроектирование). Этот процесс позволяет исключить наиболее важные риски, однако приходится жертвовать качеством конечного продукта, особенно удобством его сопровождения. Употребление термина «перепроектирование» весьма произвольно. Большинство подобных действий можно описать скорее как «втискивание» изменений и латание дыр в уже существующей реализации с целью минимизации суммарных усилий, требующихся для решения проблемы. Внесение такого рода изменений не способствует сохранению общей целостности разработки и соответствующего удобства сопровождения.

Функциональная декомпозиция, определяемая требованиями

Традиционно важнейшее значение для процесса разработки ПО имеют требования: сначала пытаются точно определить требования, а затем стремятся точно реализовать их. Такой подход зависит от того, насколько полно и недвусмысленно удается определить требования до начала выполнения других видов деятельности по разработке ПО. При этом все требования наивно рассматриваются как одинаково важные и неизменные на протяжении всего жизненного цикла разработки ПО. В действительности все эти условия редко оказываются выполненными. Определение требований является сложной и важной составляющей процесса разработки ПО. Фактически каждая большая программа сталкивается с серьезными трудностями при определении требований (см. приложение А). Более того, отношение ко всем требованиям как к одинаково важным отнимает время от разработки действительно значимых требований и заставляет выполнять лишнюю бумажную работу, связанную с отладкой, тестированием, поддержкой и т.д., — бумажную работу, которая неминуемо будет выброшена в корзину позже, по мере эволюции основных требований и соответствующего понимания проекта.

В качестве примера рассмотрим крупномасштабный проект, такой как CCPDS-R (см. приложение D), в котором требования к ПО включают в себя 2000 *пунктов*. (*Пункт* — это отдельное требование, например: «система должна выдерживать одиночные сбои аппаратуры без потери критичных возможностей».) Адекватная работа с определяющими проектными требованиями в таких системах (обычно всего от 20 до 50 пунктов) становится весьма затруднительной, если контракт налагает обязательства, чтобы все 2000 пунктов были определены и учитывались по достижении каждой основной контрольной точки разработки. Объем трудозатрат, которые могли бы быть направлены на решение действительно важных проектных проблем, распыляется на излишний багаж в 1950 пунктов, который приходится тащить за собой, и на работы, связанные с отладкой, тестированием, документацией и т.д.

Еще одним свойством традиционного подхода является то, что требования обычно специфицируются как функциональные. Процесс создания ПО по водопадной модели исходит из основополагающего допущения о том, что ПО само по себе декомпозируется на отдельные функции; требования в этом случае предъявляются к получившимся компонентам. Такая декомпозиция зачастую весьма далека от объектно-ориентированной декомпозиции и от использования уже существующих компонентов. Функциональная декомпозиция нередко закрепляется в контрактах, контрактах субподряда и в декомпозиции работ, что в значительной мере препятствует подходу, в котором ведущую роль играет архитектура. На рис. 1.4 показан результат применения подхода, в котором основную роль играют требования: здесь структура ПО организована в соответствии с определенной структурой требований.

Противостояние между участниками проекта

Для традиционного процесса характерна тенденция к возникновению противоречивых отношений между заинтересованными сторонами, что связано во многом со сложностями определения требований и с обменом информацией исключительно в виде бумажных документов, содержащих проектные данные в специальных форматах. Отсутствие жестко заданной системы записи приводит к созданию субъективных отчетов и к произвольному порядку обмена информацией.

Следующая последовательность действий являлась типичной при заключении большинства контрактов на создание ПО:

1. Исполнитель готовил черновой вариант контракта, содержавший промежуточные материалы, и отправлял его заказчику.

2. Заказчик должен был дать свои замечания (обычно в течение 15 – 30 дней).

3. Исполнитель включал эти замечания в контракт и представлял (обычно в течение 15 – 30 дней) окончательную версию на утверждение.

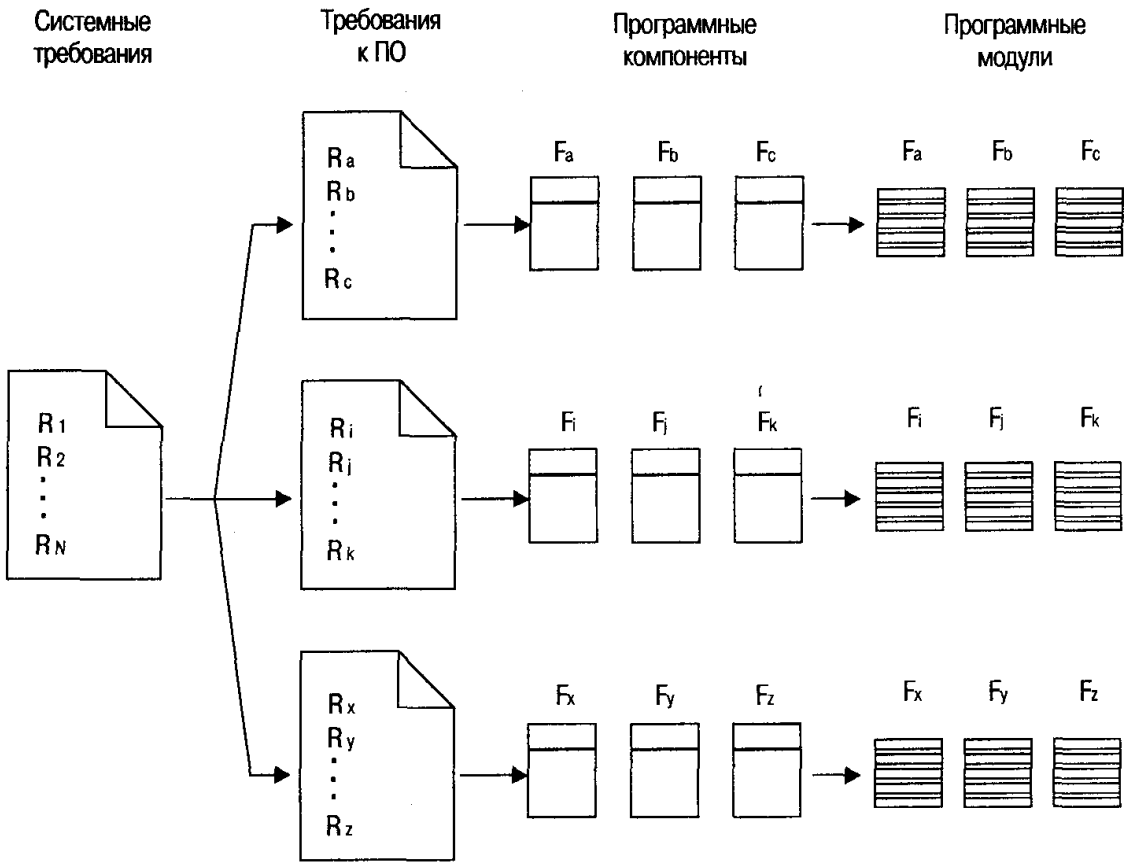


Рис. 1.4. *Неоптимальная организация программных компонентов, получающаяся при подходе, в котором ведущую роль играют требования*

Подобный процесс с одноразовым рассмотрением предполагал способность к быстрому реагированию как со стороны заказчиков, так и со стороны исполнителей. Издержки в процессе обмена бумагами оказывались недопустимыми. Подобный подход приводил даже к тому, что отношения между заказчиком и исполнителем вырождались до взаимного недоверия, что затрудняло достижение баланса между требованиями, сроками и затратами.

Чрезмерное внимание, уделяемое документации и совещаниям для обмена мнениями

При традиционном процессе важное место занимало изготовление различной документации, в которой описывался программный продукт, при этом внесению существенных изменений в сам продукт уделялось недостаточно внимания. Основные контрольные точки обычно представляли собой церемониальные совещания, которые собирались исключительно для обсуждения конкретных документов. Подрядчики были вынуждены производить буквально тонны бумаг для демонстрации прогресса заинтересованным лицам, вместо того чтобы направлять свою энергию на решение задач, которые могли бы позволить снизить риск и получить

качественное ПО. Зачастую выступающие и аудитория рассматривали простые вещи, которые им и без того были понятны, вместо действительно сложных и важных проблем. Вследствие этого большинство совещаний имело низкую ценность для разработчиков и высокую стоимость в терминах усилий и времени, затраченных на их подготовку и проведение. Они лишь создавали видимость прогресса. Результаты типичного совещания, посвященного разработке, сведены в таблицу 1.2.

Таблица 1.2.
Результаты рассмотрения разработки в традиционной модели проекта

Видимые результаты	Реальные результаты
Много выступлений перед широкой аудиторией	Только небольшой процент аудитории разбирается в ПО.
	В выступлениях и документах представлены лишь некоторые важные моменты и риски из тех, что присущи сложным программным системам.
Представление проекта как соответствующего требованиям	Не существует никаких веских свидетельств соответствия.
	Соответствие неоднозначным требованиям малоценно.
Охват требований (обычно сотни)	Очень немногие требования (десятки) непосредственно влияют на ход проекта.
	Работа со всеми требованиями не позволяет сосредоточиться на действительно важных требованиях.
Проект считается «непорочным, пока не доказано обратное»	Проект всегда имеет пороки.
	Недостатки проекта проявятся на более поздних этапах жизненного цикла.

Диагностика пяти симптомов (см. выше) проектов, предрасположенных к появлению проблем, может оказаться весьма сложной, особенно на ранних стадиях жизненного цикла, когда проблемы, связанные с традиционным подходом, могли бы быть решены наиболее просто. Соответственно, современные проекты по созданию ПО должны использовать механизмы, которые дают возможность оценить состояние здоровья проекта на ранних стадиях его жизненного цикла и которые позволяют периодически проводить объективные проверки.

1.2 Эффективность традиционного управления проектами

Одностраничная статья Барри Боэма «Industrial Software Metrics Top 10 List» («Список 10 главных правил промышленного создания ПО») [Boehm, 1987] дает объективную характеристику положению дел с разработкой ПО. (Существует мало свидетельств того, чтобы за последнее

десятилетие произошли какие-либо значительные изменения.) Хотя многие из этих правил являются довольно общими, они точно описывают некоторые фундаментальные экономические отношения, сложившиеся при использовании традиционного процесса создания ПО за последние 30 лет.

Ниже *курсивом* выделяются правила Боэма и обычным шрифтом даются мои комментарии.

1. *Поиск и обнаружение ошибки в ПО после его сдачи обходится в 100 раз дороже, чем поиск и обнаружение ошибки на ранних стадиях разработки.*

   ▲ Эта оценка является главным обоснованием для изменения процесса практически любого направления. Она не является уникальной для разработки ПО. Если, например, некая крупная автомобильная компания исправляет дефект, обнаруженный после продажи, стоимость ремонта может быть на много порядков выше, чем стоимость выявления дефекта на стадии разработки или производства.

2. *Можно сократить срок разработки ПО на 25% от номинального, но не более.*

   ▲ Одной из причин такого положения вещей является то, что для сокращения времени разработки на N% требуется увеличить количество персонала на M% (в предположении, что другие параметры остаются неизменными). Любой рост числа работников требует увеличения затрат на управление. Вообще говоря, допустимые отклонения для этих затрат с учетом графика выполнения параллельных работ, замораживания последующих работ и других ресурсных ограничений составляют около 25%. Оптимальным для работы, требующей 100 человеко-месяцев, может быть выполнение ее за 10 месяцев 10-ю работниками. Смогут ли такую работу выполнить 100 человек за один месяц? 50 человек за два месяца? А может быть, 20 человек за 5 месяцев? Понятно, что это все нереальные варианты. Правило 25%-ного сокращения гласит, что в этом случае допустимое предельное значение — 7.5 месяцев (хотя существует вероятность того, что потребуются дополнительные человеко-месяцы, возможно, около 20). Любое дальнейшее сокращение сроков работ обречено на неудачу. С другой стороны, оптимальный график работ может растягиваться почти произвольно и в зависимости от сотрудников может выполняться за существенно большее время при меньшем штате. Например, если отводится 25 месяцев на выполнение работ, то вполне можно уложиться в 75 человеко-месяцев при использовании трех человек.

3. *На каждый доллар, вложенный в разработку, приходится тратить два доллара на сопровождение.*

   ▲ Боэм называет это правило «железным законом разработки ПО». Независимо от того, создается ли продукт долговременного использования, у которого выходят по две коммерческие версии в год, или единственное в своем роде ПО для конкретного заказчика, количество денег, затрачиваемых на весь цикл сопровождения, будет с некоторой вероятностью в два раза больше суммы, истраченной за весь цикл разработки. На первый взгляд трудно

понять, хорошее это соотношение или плохое. Для сектора коммерческих продуктов оно определяется прежде всего коммерческим успехом на рынке. Успешные программные продукты (такие, как Oracle, приложения Microsoft, Rational Rose и операционная система UNIX) существуют в течение очень долгого времени, что может приводить к более высокому значению отношения стоимости сопровождения к стоимости разработки. С другой стороны, менеджеры проектов по созданию «одноразового» ПО редко планируют большие затраты на его сопровождение. В любом случае каждому, кто был занят в индустрии ПО в течение последних 10 – 20 лет, известно, что ПО всегда трудно сопровождать.

4. *Стоимость разработки и сопровождения ПО является прежде всего функцией числа строк исходного кода.*

   ▲ Это правило является следствием преобладания разработки ПО на заказ, отсутствия коммерчески используемых компонентов и отсутствия повторного применения, что было присуще эре традиционного процесса.

5. *Различия в уровне разработчиков приводят к огромной разнице в продуктивности при создании ПО.*

   ▲ Это самая главная мудрость традиционного процесса: нанимайте хороших работников. Данному правилу зачастую придается как недостаточное, так и чрезмерное значение. Когда отсутствуют объективные сведения относительно причин успеха или неудачи, очевидным козлом отпущения становится квалификация персонала. Это суждение субъективно, и его трудно оспаривать.

6. *Общее отношение стоимости ПО к стоимости аппаратуры продолжает расти. В 1955 г. оно составляло 15:85; в 1985 г. – 85:15.*

   ▲ Тот факт, что на ПО приходится 85% стоимости большинства систем, является не столько следствием продуктивности создания ПО (которое к тому же оказывается не столь хорошим, как хотелось бы), сколько следствием того уровня функциональных задач, выполнение которых возлагается на ПО при принятии решений о системе в целом. Нужда в программном обеспечении, широта его использования и его сложность продолжают безгранично расти.

7. *При создании ПО всего лишь около 15% усилий затрачивается собственно на программирование.*

   ▲ Это весьма важный показатель необходимого равновесия. Для успешного осуществления проекта приходится помимо кодирования выполнять много других задач. Управление требованиями, проектирование, тестирование, планирование, контроль за ходом проекта, управление изменениями и подготовка инструментария — одинаково важные задачи, на которые тратится приблизительно 85% ресурсов.

8. *Программные системы и продукты обычно стоят в три раза дороже в пересчете на одну строку исходного кода, чем отдельные программы. Продукты, состоящие из программных систем (т.е. системы систем), дороже в девять раз.*

▲ Эта экспоненциальная зависимость лежит в основе того, что называют платой за масштаб (diseconomy of scale). В отличие от других товаров, чем больше объем создаваемого ПО, тем дороже оно обходится в пересчете на строку исходного кода.

9. *Сквозной контроль позволяет обнаружить 60% ошибок.*

▲ Возможно, это и так. Однако с учетом правила 1 сквозной контроль не позволяет обнаруживать значимые ошибки, особенно на ранних стадиях жизненного цикла. Дефект дефекту рознь. Вообще говоря, сквозной контроль и другие формы контроля, производимого человеком, хороши для обнаружения ошибок, лежащих на поверхности, и проблем, касающихся стиля. При использовании для разработки специальной системы записи просмотр человеком может служить первичным механизмом гарантии качества, но он не позволяет обнаруживать проблемы второго, третьего, N-ого порядка, такие как конфликты ресурсов, узкие места, связанные с производительностью, конфликты управления и т.п. Более того, очень немногие люди способны находить даже семантические ошибки первого порядка в тексте программы. Скольким программистам удается компилировать свою собственную программу с первого раза?

10. *80% работы выполняют 20% работающих.*

▲ Это основополагающее утверждение остается верным практически для любых дисциплин, связанных с разработкой (или для любой профессиональной дисциплины, если уж на то пошло). Я расширил это правило. Следующие фундаментальные постулаты лежат в основе объяснения современного подхода к процессу управления созданием ПО:

80% разработки выполняется для удовлетворения 20% требований

80% процентов стоимости ПО приходится на 20% компонентов

80% ошибок возникают по вине 20% компонентов

80% ПО выбрасывается и заново переделывается из-за 20% ошибок

80% ресурсов расходуются на 20% компонентов

80% разработок выполняются с помощью 20% инструментария

80% успеха обеспечивают 20% людей

Эти соотношения дают хорошую точку отсчета для поиска способов усовершенствования процесса и технологии. Они представляют приблизительные практические правила, которые объективно характеризуют ход традиционного процесса управления созданием ПО и традиционные технологии. В последующих главах я буду возвращаться к этим соотношениям для того, чтобы обосновать новый подход, защитить старый подход и определить количественные характеристики процесса или усовершенствований технологии.

Глава 2

Эволюция экономики разработки ПО

Ключевые моменты

- Экономические результаты традиционных проектов по созданию ПО являются следствием преобладания разработки на заказ, узкоспециализированных процессов и платы за большой масштаб.
- Современные модели стоимости основываются прежде всего на эмпирических данных проектов, среди которых мало случаев успешной итерационной разработки.
- Трудно получить хорошие оценки стоимости ПО. Ответственным за принятие решений приходится иметь дело с очень неточными оценками.
- Современная схема процесса позволяет бороться с основными причинами платы за большой масштаб, присущей традиционному процессу создания ПО.

Программная инженерия в основном представляет собой интеллектуальный вид деятельности, направленный на решение проблем высочайшего уровня сложности с бесконечным числом неизвестных в постоянно меняющихся условиях. Подходы к созданию ПО 60-х и 70-х гг. лучше всего описываются термином «кустарное производство», когда в каждом проекте использовались собственный процесс и собственный инструментарий. В 80-х и 90-х гг. индустрия создания ПО перешла в разряд инженерной дисциплины. Однако большинство проектов по созданию ПО этого периода по-прежнему предполагало проведение интенсивных исследований, которым были свойственны творческий подход и плата за большой масштаб. Следующее поколение процессов создания ПО, в особенности методы, представленные в данной книге, движется в сторону подхода с более интенсивным производством, которому свойственны автоматизация и экономия при больших масштабах.

2.1 ЭКОНОМИКА ПО

Большинство моделей для определения стоимости ПО может быть сведено к функции пяти основных параметров: размера, процесса, персонала, среды и требуемого качества.

1. *Размер* конечного продукта (для компонентов, написанных вручную), который обычно измеряется числом строк исходного кода или количеством функциональных точек, необходимых для реализации данной функциональности.
2. Особенности *процесса*, используемого для получения конечного продукта, в частности его способность избегать непроизводительных видов деятельности (переделок, бюрократических проволочек, затрат на взаимодействие).
3. Возможности *персонала*, участвующего в разработке ПО, в особенности его профессиональный опыт и знание предметной области проекта.
4. *Среда*, которая состоит из инструментов и методов, используемых для эффективной разработки ПО и автоматизации процесса.
5. Требуемое *качество* продукта, что включает в себя его функциональные возможности, производительность, надежность и адаптируемость.

Соотношение между рассчитываемой стоимостью и этими параметрами может быть записано следующим образом:

$$\text{Трудоемкость} = (\text{Персонал})(\text{Среда})(\text{Качество})(\text{Размер}^{\text{Процесс}})$$

Для оценки стоимости ПО создано несколько параметрических моделей; все они, вообще говоря, могут быть сведены к такой форме. Один из важных аспектов экономики создания ПО (как это представляется в современных моделях определения стоимости ПО) заключается в том, что связь между работой и размерами определяет плату за большой масштаб. Плата за большой масштаб при разработке ПО является результатом того, что показатель экспоненты процесса больше единицы. В отличие от большинства производственных процессов, чем больше ПО создается, тем дороже оно обходится в пересчете на одну единицу. Например, для некоторого произвольного приложения программное решение объемом в 10 000 строк обойдется дешевле в пересчете на одну строку, чем программное решение объемом 100 000 строк. На сколько дешевле? Предположим, что для создания 100 000-строчной системы требуется 900 человеко-месяцев, или около 111 строк за один человеко-месяц, или 1.37 часа на одну строку. Если бы та же самая система состояла из 10 000 строк при неизменных остальных параметрах, то проект оценивался бы приблизительно в 62 человеко-месяца, или 175 строк за один человеко-месяц, или 0.87 часа на одну строку. (На рис. B.1 в приложении B дается более подробное описание этого примера с использованием модели оценки стоимости COCOMO.) Стоимость одной строки для меньшего приложения

оказывается гораздо ниже таковой для большего приложения. Причина этого заключается прежде всего в сложности управления межличностными взаимодействиями по мере того, как число членов команды (и соответственно число целей, условий их достижения, технических предпочтений) возрастает. Эта плата за большой масштаб характерна для любого исследовательского проекта, продуктом которого является единственный в своем роде объект интеллектуальной собственности.

Рис. 2.1. *Три поколения экономики создания ПО, ведущие к намеченной цели*

На рис. 2.1 показаны три поколения основных достижений технологии в части инструментария, компонентов и процессов. Необходимый уровень качества и персонал принимаются постоянными. По оси ординат откладывается стоимость единицы ПО (выберите, какая вам больше нравится: строка исходного кода (SLOC), функциональная точка, компонент), созданного некоей организацией. Ось абсцисс представляет жизненный цикл использования ПО в бизнесе данной организации. Три поколения процессов разработки ПО могут быть определены следующим образом:

1. *Традиционный:* 60-е — 70-е гг., кустарное производство. Организации используют кустарный инструментарий, кустарные процессы и практически все компоненты для заказчика пишутся на примитивных языках. Результат выполнения проекта был легко предсказуем в том смысле, что он практически никогда не укладывался в заранее заданные стоимость, сроки и качество.

2. *Переходный:* 80-е — 90-е гг., программная инженерия. Организации используют воспроизводимые процессы и готовые инструменты, а большинство создаваемых компонентов (>70%) пишется на языках высокого уровня. Некоторые компоненты (<30%) становятся доступными в качестве коммерческого продукта, включая операционные системы, системы управления базами данных, сетевое ПО и графический интерфейс пользователя. В течение 80-х гг. некоторые организации начинают достигать экономии при больших масштабах, однако с ростом сложности приложений (особенно при переходе к распределенным системам) существовавшие языки, методы и технологии оказались недостаточными для того, чтобы поддерживать требуемый уровень промышленного создания.

3. *Современная практика:* начиная с 2000 г., производство ПО. Философия этой книги заключается в применении управляемых и измеряемых процессов, интегрированных сред автоматизации и по большей части (70%) готовых компонентов. Возможно, всего лишь 30% компонентов следует создавать на заказ. Используя преимущества технологии создания ПО и интегрированных сред разработки, можно очень быстро создавать системы, построенные из компонентов.

Технологии, позволяющие автоматизировать среду разработки, уменьшить размер ПО и усовершенствовать процесс, не являются независимыми. Для каждого нового периода времени ключевым становится некоторое совершенствование всех технологий. Например, преимущества нового процесса не могут быть успешно использованы без новых технологий создания компонентов и повышения степени автоматизации.

Переход к современной практике и надежда на улучшение экономики создания ПО ни в коем случае не являются гарантированными. Мы должны оставаться реалистами, сравнивая надежды, которые дает хорошо разработанный процесс следующего поколения, с уродливыми реалиями истории. Я готов держать пари, что многие из тех организаций, которые попытаются выполнить проекты, используя современные методы и технологии, придут в результате к той же старой доброй неразберихе.

Рис. 2.2. *Возврат инвестиций в различных областях применения*

Организации достигают большей экономии при больших масштабах в течение технологически успешных периодов – в рамках очень больших проектов (системы систем), продуктов долговременного использования и продуктовых линий, включающих в себя множество однотипных проектов. Рис. 2.2 дает общее представление о том, каким образом можно достичь соответствующего вида кривой возврата инвестиций (ROI, Return On Investment) при последовательных усилиях в течение всего жизненного цикла для различных областей применения.

2.2 ПРАКТИЧЕСКАЯ ОЦЕНКА СТОИМОСТИ ПО

Одной из важных проблем при оценке стоимости ПО является отсутствие хорошо документированных практических примеров проектов, в которых применялась итерационная разработка. Хотя авторы моделей оценки стоимости и заявляют, что их инструментарий пригоден для оценки проектов, использующих итерационную разработку, лишь немногие из них основываются на эмпирических данных проектов, в которых итерационная разработка была успешной. Более того, поскольку индустрия ПО оперирует противоречивыми метриками и основными единицами измерения, то данные по конкретным проектам оказываются весьма подозрительными с точки зрения их непротиворечивости и возможности сравнения. Сбор однородных данных по проекту в рамках одной организации оказывается довольно сложным; чрезвычайно сложен сбор однородных данных по разным организациям, использующим различные процессы, языки, подходы и т.д. Например, фундаментальное понятие — единица измерения размера (строка исходного кода или функциональная точка) — вычисляется везде по-разному. Кажется удивительным, что стандарты современных языков (таких, как Ada 95 и Java) не имеют определения понятия строки исходного кода для подсчета их компилятором. То, какое именно определение (функциональной точки или строки исходного кода(SLOC)) будет применено, не столь важно, точно так же, как конкретная длина фута или метра может быть абсолютно произвольной. Необходимо лишь, чтобы все пользовались одним и тем же определением.

Среди разработчиков и поставщиков моделей и средств для оценки стоимости ПО давно идут различные споры. Для нас практический интерес представляют три темы этих споров:

1. Какую модель оценки стоимости ПО следует использовать.
2. Следует ли измерять объем ПО в строках исходного кода или в функциональных точках.
3. Что может считаться хорошей оценкой.

В индустрии ПО между собой конкурируют около 50 поставщиков средств, данных и услуг по оценке стоимости ПО. Известны общедоступные модели и средства для оценки стоимости ПО (такие, как COCOMO, CHECKPOINT, ESTIMACS, KnowledgePlan, Price-S, ProQMS, SEER, SLIM, SOFTCOST и SPQR/20), а также огромное количество моделей, применяемых в конкретных организациях. Поскольку мой первый опыт использования этих моделей был связан с COCOMO и ее производными — Ada COCOMO и COCOMO II, именно эти модели служат основой моих аргументов и взглядов на перспективы развития экономики ПО. Кроме того, COCOMO является одной из наиболее открытых и хорошо документированных моделей оценки стоимости. Процесс превращения COCOMO в современную версию — COCOMO II — описан в приложении B. Некоторые части этого приложения не могут напрямую использоваться в сегодняшних методах и технологиях, тем не менее оно дает интересную ретроспективу эволюции проблем и приоритетов экономики ПО за последние 20 лет.

Много споров вызывает вопрос об измерении объема ПО. На самом деле существуют две основные точки зрения: строки исходного кода и функциональные точки. Эти возможности доказали свою большую полезность, по сравнению с третьей, которая является субъективной или узкоспециализированной точкой зрения, используемой отдельными не слишком зрелыми организациями, которым не приходится проводить систематическое измерение объема ПО.

Многие эксперты в области ПО утверждают, что SLOC — плохая единица измерения. Однако когда говорят, что сегмент программы содержит 1000 строк исходного кода, большинство людей способно представить себе ее общий размер. В случае же если программа описывается в терминах 20 функциональных точек, 6 классов, 5 вариантов использования, 4 объектных точек, 6 файлов, 2 подсистем, 1 компонента или 6000 байтов, то многие, включая экспертов ПО, начнут задавать дополнительные вопросы для того, чтобы получить представление об описываемом коде. (И многие спросят, а сколько же в ней SLOC.) Таким образом, SLOC оказывается единственной единицей измерения, которая до сих пор обладает определенной ценностью.

Десять лет тому назад я являлся ярым сторонником SLOC как единицы измерения, поскольку она хорошо работала в приложениях, которые создавались преимущественно на заказ, и поскольку измерения в SLOC было легко автоматизировать и осуществлять на практике. Сегодня же возможности современных языков и применение компонентов, автоматическая генерация исходного кода и ориентация на объекты сделали SLOC неточной единицей измерения. В приложении D в качестве характерного примера описываются тщательно проработанные подходы для подсчета SLOC с тем, чтобы они позволяли учитывать повторное использование, разработку на заказ и инструментарий для генерации кода в рамках большого проекта по созданию ПО.

Применение функциональных точек имеет много последователей, включая Каперса Джонса, который указывает на сложности, связанные с использованием SLOC для объектно-ориентированных программ [Jones, 1994]. Международный консорциум по использованию функциональных точек (the International Function Point User's Group), образованный в 1984 г., является доминирующей ассоциацией по вопросам измерения ПО. Самым главным преимуществом применения функциональных точек является то, что этот метод не зависит от конкретной технологии и, таким образом, предоставляет элементарные единицы измерения для сравнения различных проектов и организаций. Основной его недостаток заключается в том, что определения абстрактны, а способ проведения измерений не вытекает непосредственно из входящих в него положений.

Оба способа измерения имеют свои недостатки, но я думаю, что можно пользоваться любым из них. Выполнить хоть какие-нибудь измерения лучше, чем не делать ничего. Каждому, кто пытается сравнивать разные проекты или разные организации, в качестве единиц измерения следует использовать функциональные точки. Кроме того, функциональные точки являются, вероятно, более точным способом измерения на ранних стадиях жизненного цикла проекта. На поздних же стадиях более

полезной и более точной основой для различных измерений становится SLOC. В главе 16 представлена моя гипотеза о модели стоимости следующего поколения, которая могла бы позволить минимизировать или даже отказаться от необходимости проводить измерения в SLOC.

Общая точность традиционных моделей стоимости (таких, как COCOMO) описывается как «в пределах 20% по стоимости, 70% по времени». Такой уровень непредсказуемости традиционного процесса разработки ПО отпугнет любого инвестора, особенно в свете того факта, что некоторые проекты не подтвердили своей оценки, а оказались лучше, чем ожидалось. Это интересное явление следует учитывать при планировании трудоемких работ. До тех пор пока не появляются дополнительные стимулы для опережения общего графика работ, проекты редко выполняются быстрее, чем планировалось. Почему? Команды и отдельные работники составляют собственные планы для выполнения своих задач. И если временные рамки оказываются не слишком жесткими, то они либо тратят свою энергию на посторонние занятия (дополнительное обучение, помощь другим или валяние дурака), либо продолжают повышать качество сверх необходимого уровня. Исполнители практически никогда не предлагают сократить сроки. Но даже если они и выступят с подобным предложением, то, вероятнее всего, наткнутся на сопротивление других сотрудников, с которыми им следует синхронизировать свою работу. Поэтому планы должны быть настолько амбициозными, насколько это возможно.

На самом деле, в большинстве случаев стоимостные модели используются «от противного» (для подтверждения объявленной стоимости), а вовсе не по прямому назначению (для определения той цены, которую следовало бы запросить). На рис. 2.3 показана обычная практика: менеджер проекта сначала определяет, какую цену следует объявить, а затем манипулирует параметрами и размерами до тех пор, пока не удается эту цену обосновать. Обоснованием объявленной цены может быть стремление выиграть тендер, выбить у заказчика финансирование, добиться финансирования внутри организации либо достигнуть каких-либо других целей.

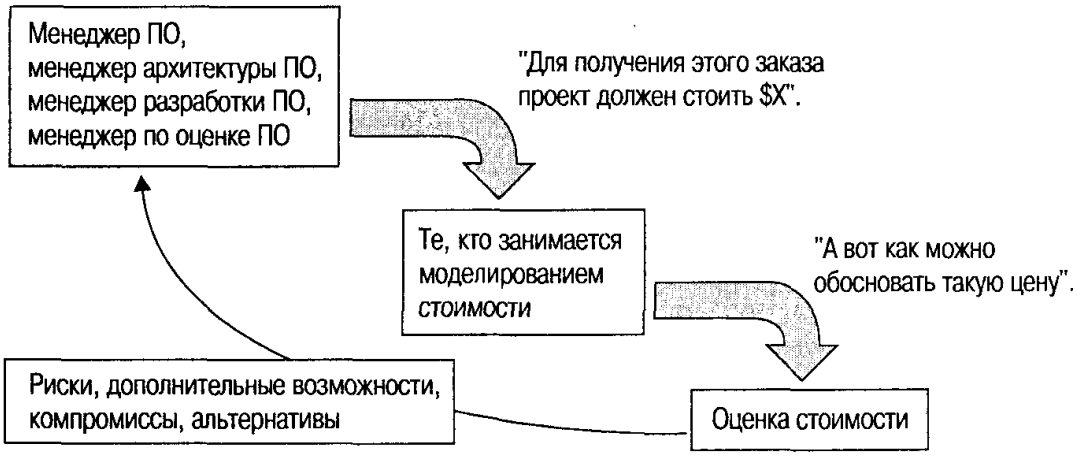


Рис. 2.3. *Обычный процесс определения стоимости*

Процесс, представленный на рис. 2.3, вовсе не так уж плох. На самом деле, он абсолютно необходим для проведения анализа стоимостных рисков и для объективного понимания чувствительности модели и степени влияния различных факторов. Он заставляет менеджера проекта тщательно изучать риски, связанные с достижением объявленной стоимости, и обсуждать эту информацию с остальными заинтересованными сторонами. В результате обычно предлагается внесение различных изменений в планы, разработку, процесс или намерения. Данный подход предоставляет хорошее обоснование для оценки стоимости и общего ее анализа.

Практика в этой области такова, что независимые оценки стоимости (т.е. выполненные людьми, которые никак не зависят от команды разработчиков) обычно неточны. Единственным способом, позволяющим получить заслуживающую доверия оценку, является следующий: компетентная команда — менеджер проекта вместе с менеджерами по созданию архитектуры, разработке и тестированию — выполняет несколько итераций по оценке стоимости и анализу чувствительности модели. Для того чтобы проект мог быть успешно выполнен, эта команда должна признать свое авторство произведенной оценки стоимости.

Из чего состоит хорошая оценка стоимости ПО? Этот непростой вопрос подробно обсуждается в главе 10. Если же говорить кратко, то у хорошей оценки должны быть следующие атрибуты:

- Она создается и поддерживается менеджером проекта, командой по разработке архитектуры, командой разработчиков и командой, выполняющей тестирование, т.е. теми, кто несет ответственность за выполнение работ.
- Она воспринимается всеми исполнителями как амбициозная, но выполнимая.
- Она базируется на подробно описанной модели стоимости ПО, имеющей заслуживающее доверия основание.
- Она основывается на данных по аналогичным проектам, которые включают в себя аналогичные процессы, аналогичные технологии, аналогичную среду, аналогичные требования к качеству и аналогичную квалификацию работников.
- Она подробно описывается таким образом, чтобы все ключевые области риска были хорошо видны, а вероятность успеха оценивалась объективно.

Идеальную оценку можно найти путем экстраполяции хорошей оценки, полученной на основе устоявшейся модели стоимости и использующей опыт выполнения ряда аналогичных проектов той же командой, которая применяла те же зрелые процессы и инструментарий. Такая ситуация на практике встречается редко. Когда команда приступает к осуществлению нового проекта, хорошие оценки могут быть получены обычным путем на более поздних этапах жизненного цикла зрелого проекта, использующего зрелый процесс.

Глава 3

Совершенствование экономики разработки ПО

Ключевые моменты

- Современная технология разработки ПО позволяет строить системы с меньшим количеством строк исходного кода, создаваемого человеком.
- Современные процессы создания ПО являются итерационными.
- Современные среда разработки и среда сопровождения представляют собой механизм для автоматизации процесса.

Улучшения экономики разработки ПО трудно достигнуть и доказать. Обсуждение этой темы в учебниках по программированию, рекламных журналах и литературе по программным продуктам кишит непонятными словами, противоречивыми единицами измерения, несогласием экспертов друг с другом и бесконечными преувеличениями. Если мы попытаемся рассмотреть любой из аспектов, касающихся улучшения экономики ПО, то придем к заключениям, имеющим весьма ограниченное значение, и к наблюдениям весьма невысокой ценности. Аналогично, если организация слишком сосредоточивается на улучшении какого-либо одного аспекта процесса разработки ПО, она вряд ли сможет получить сколько-нибудь значительный экономический эффект даже в том случае, если ей удастся существенно улучшить этот самый аспект.

Ключом к осязаемому улучшению может служить одновременное наступление по нескольким взаимосвязанным направлениям. Я сгруппировал самые важные направления вокруг пяти основополагающих параметров модели стоимости ПО, представленной в главе 2.

1. Уменьшение *размера* или *сложности* того, что предстоит разрабатывать
2. Усовершенствование *процесса* разработки

3. Использование более квалифицированного *персонала* или хороших команд (что необязательно означает одно и то же)
4. Использование лучшей *среды* (инструментария для автоматизации процесса)
5. Достижение уступок и компромиссов для пороговых значений *качества*

Эти параметры приведены в порядке значимости для большинства видов ПО. В таблице 3.1 перечислены некоторые усовершенствования в технологии, шаги по повышению эффективности процесса и управленческие подходы, направленные на улучшение экономики разработки и интеграции ПО.

Таблица 3.1.
Важные тенденции совершенствования экономики ПО

Параметры модели стоимости	Тенденции
Размер Технологии разработки, основанные на абстракции и компонентном подходе	Языки высокого уровня (C++, Ada 95, Java, Visual Basic и др.) Объектно-ориентированный анализ, проектирование и программирование Повторное использование Коммерческие компоненты
Процесс Принципы и методы	Итерационная разработка Модели зрелости процесса Упреждающая разработка архитектуры Реформа приобретения
Персонал Человеческие факторы	Обучение и совершенствование навыков персонала Командная работа Дух победителей
Среда Автоматизация технологий и инструментов	Интегрированный инструментарий (визуальное моделирование, компилятор, редактор, отладчик, управление изменениями и т.д.) Открытые системы Быстродействие аппаратной платформы Автоматизация кодирования, документирования, тестирования, анализа
Качество Быстродействие, надежность, точность	Быстродействие аппаратной платформы Оценка на основе демонстраций Статистический контроль качества

Большинство экспертов ПО заметят тесную взаимозависимость между этими направлениями. Например, инструментарий позволяет уменьшить размер и улучшить процесс. Уменьшение размера привносит изменения в сам процесс, а совершенствование процесса изменяет требования к

инструментарию. Рассмотрим ПО пользовательского интерфейса. Лет двадцать тому назад командам, разрабатывающим пользовательский интерфейс, пришлось бы потратить много времени на анализ операций, экранных форм, динамики изменения экранного представления и на учет человеческого фактора. Все это приходилось выполнять на бумаге, поскольку перевод разработок в форму выполняемого кода, даже в виде неформальных прототипов, оказывался чрезвычайно дорогим. Таким образом, в этом процессе особое внимание уделялось тяжеловесному набору «бумажных» материалов, полученных на ранних этапах разработки, а также достижению согласия с пользователями для того, чтобы эти «требования» могли быть заморожены и чтобы можно было минимизировать высокую стоимость разработки.

Технология графического пользовательского интерфейса (GUI) является хорошим примером инструментария, обеспечивающего применение нового и измененного процесса. По мере того как технология GUI совершенствовалась, традиционный процесс взаимодействия с пользователем устаревал. Инструментарий для построения GUI позволяет командам разработчиков создавать готовый пользовательский интерфейс быстрее и с меньшей стоимостью. Теперь нет необходимости в описаниях на бумаге; фактически они являлись препятствием на пути к повышению эффективности процесса. Анализ операций и человеческого фактора по-прежнему остается важным, но теперь это можно делать в той реальной среде, в которой продукт будет использоваться, с применением уже существующих примитивов и строительных блоков. Циклы разработки и доработки, которые обычно тянулись месяцами, теперь могут выполняться за нескольких дней или недель. Старый процесс позволял лишь убедиться в том, что пользовательский интерфейс полностью проанализирован и разработан, поскольку в рамках проекта мог осуществляться только один цикл создания. Новый процесс направлен на проверку пользовательского интерфейса посредством создания нескольких реальных версий с постоянным внесением изменений, возникающих в результате обратной связи с пользователем, и на достижение устойчивого понимания требований и проблем разработки во взаимном равновесии.

Можно утверждать, что преимущества процесса (такие, как необходимость итераций и возможность экспериментирования с пользовательскими интерфейсами) привели к развитию инструментария, или наоборот, преимущества технологии привели к изменению процесса. Скорее всего, верно и то, и другое. Суть заключается в том, что все пять основных параметров уравнения оценки стоимости не являются ни взаимоисключающими, ни независимыми друг от друга. Они взаимосвязаны.

Другим важным фактором, который оказал существенное влияние на улучшение технологии ПО, является постоянно растущее быстродействие аппаратуры. Большая тактовая частота, большая память и более высокая пропускная способность ликвидировали многие источники сложностей в реализации ПО. Стали возможны грубые решения «в лоб», а усовершенствование аппаратуры, вероятно, является тем преимуществом, на котором основывается большинство значимых улучшений технологии ПО.

3.1 УМЕНЬШЕНИЕ РАЗМЕРА ПРОГРАММНОГО ПРОДУКТА

Наиболее действенным способом, повышающим возврат инвестиций (ROI), обычно является создание продукта, который достигает выполнения поставленных перед ним задач с помощью минимального количества исходного материала, написанного человеком. В данном случае *компонентно-ориентированная разработка* (component-based development) представлена как общий способ уменьшения размера программы на «исходном» языке программирования, который необходим для решения поставленных задач. Повторное использование, объектно-ориентированная технология, автоматическая генерация кода и языки программирования более высокого уровня – все это направлено на создание требуемой системы с меньшим количеством операторов исходного кода, написанных человеком. Уменьшение размера – главная движущая сила применения языков программирования более высокого уровня (таких, как C++, Ada 95, Java, Visual Basic и языков четвертого поколения), автоматических генераторов кода (CASE-средств, средств визуального моделирования, построителей GUI), повторного использования коммерческих компонентов (операционных систем, сред для работы с окнами, систем управления базами данных, промежуточного ПО, сетей) и объектно-ориентированных технологий (UML, средств визуального моделирования, архитектурных решений).

При обсуждении вопроса об уменьшении размера программного продукта необходимо сделать одно предостережение. Кажется очевидной рекомендация: отсутствующий код не надо разрабатывать, и он не будет служить источником ошибок. Однако это не совсем так. Вообще говоря, применение технологий уменьшения размера приводит лишь к снижению количества исходных строк, написанных человеком, но при этом обычно увеличивается объем кода, выполняемого компьютером. Поэтому первая часть рекомендации является правильной, а вторая часть правильной может и не быть. Опыт многих проектных команд позволяет сделать вывод, что устоявшиеся и надежные технологии уменьшения размера чрезвычайно полезны для получения экономических выгод. Недоработанные технологии, возможно, и уменьшат количество кода, однако при этом потребуют таких вложений для достижения необходимого уровня качества и быстродействия, что влияние, которое они окажут на ход выполнения проекта в целом, окажется отрицательным.

3.1.1 Языки

Универсальные функциональные точки (UFP) являются полезными единицами измерения для получения оценок, независимых от языка программирования, на ранних стадиях жизненного цикла. Основными составляющими функциональных точек являются входные и выходные пользовательские данные, внутренние логические группы данных, внешние интерфейсы по данным и внешние запросы. После того как сформулировано предварительное решение и выбран язык разработки,

полезными единицами измерения для оценки объема ПО становятся SLOC – строки исходного кода. Важные соотношения между SLOC и функциональными точками приводятся в работе [Jones, 1995]. Некоторые из этих соотношений представлены в таблице 3.2.

Таблица 3.2.
Языковая выразительность некоторых из наиболее популярных современных языков

Язык	Количество SLOC на одну UFP
Ассемблер	320
C	128
FORTRAN 77	105
COBOL 85	91
Ada 83	71
C++	56
Ada 95	55
Java	55
Visual Basic	35

Эти данные хорошо иллюстрируют причину заинтересованности в таких современных языках, как C++, Ada 95, Java и Visual Basic: они обладают весьма привлекательным уровнем выразительности. Однако применять эти данные следует осторожно, поскольку существует большая вероятность неправильного их употребления. Я убежден, что они отражают важные соотношения, но цифры кажутся слишком уж точными. (Без сомнения, они представляют собой среднее нескольких приблизительных значений.) Каждый язык имеет свою область применения. Visual Basic оказывается чрезвычайно выразительным и мощным при построении простых интерактивных приложений, но вряд ли будет разумным использовать его при создании бортовой программы для авиации, выполняющейся в реальном времени. Аналогично, Ada 95 может оказаться наилучшим языком для системы управления атомной электростанцией, результатом сбоя которой может стать атомная катастрофа, но вряд ли он подойдет для сильно распараллеленной, перемалывающей горы чисел научной задачи, выполняемой на суперкомпьютере. Подобные данные, имеющие отношение к индустрии ПО и распространяющиеся на различные области применения, корпорации и поколения технологий, должны интерпретироваться и использоваться с величайшей осторожностью.

Эти данные позволяют сделать два интересных наблюдения, касающихся отличий и связей между Ada 83 и Ada 95, с одной стороны, и между C и C++, с другой. Интерес Министерства обороны США к развитию языка Ada 83 частично объяснялся желанием повысить его выразительность. (Среди других причин были надежность, поддержка программирования задач в реальном времени, простота сопровождения и

гарантированный ROI за счет стандартизации языка.) Серьезной экономической мотивацией была возможность создавать программу, содержащую существенно меньшее число строк кода, чем требовалось при использовании в качестве альтернативы традиционных языков программирования – FORTRAN, COBOL, C и ассемблера. В язык Ada было включено огромное количество технологических возможностей для разработки ПО, среди которых управление конфигурацией на уровне языка, отделение интерфейса от реализации, примитивы управления архитектурой, инкапсуляция, поддержка параллелизма и многое другое. Ada 95 представляет собой хорошо спланированное расширение языка, направленное на приведение его в соответствие с новыми технологиями и на учет уроков, извлеченных из приложений в разных областях. Различие в выразительности между Ada 83 и Ada 95 заключается в основном в тех особенностях, которые были введены в язык с целью поддержки объектно-ориентированного программирования. Таким образом, в первом приближении ценность использования объектно-ориентированного программирования заключается в том, что оно позволяет писать программы, содержащие на 30% меньше строк исходного кода.

Еще более глубокое различие существует между языками C и C++. C++ включает в себя некоторые (хотя и не все) из преимуществ языка Ada, например поддержку объектно-ориентированного программирования. С другой стороны, язык C++ разрабатывался так, чтобы поддерживать язык C как собственное подмножество. Такой подход имеет свои «за» и «против». Совместимость с языком C позволила программистам, работающим на C, легко перейти на язык C++. Но обратной стороной оказывается известное в индустрии ПО явление: существует значительное количество программистов, использующих компилятор C++, но при этом программирующих на C-подмножестве, что не позволяет им достигать выразительности объектно-ориентированного языка C++. Развитие языка Java позволило решить многие проблемы C++ (в частности, исторически сложившуюся поддержку языка C, которая вдохновляла на определенные опасные программистские трюки), сохранив при этом объектно-ориентированные возможности и добавив переносимость и распределенность.

Универсальные функциональные точки могут использоваться для определения относительных размеров программ, необходимых для реализации той или иной функциональной возможности. Например, для создания некоторого приложения с заранее известными функциональными возможностями может потребоваться:

1 000 000 строк на ассемблере

400 000 строк на языке C

220 000 строк на языке Ada 83

175 000 строк на языке Ada 95 или C++

Эти значения показывают относительную выразительность различных языков. Коммерческие компоненты и автоматические генераторы кода (такие, как CASE-средства и построители GUI) способны обеспечить

еще большее снижение объема кода, генерируемого вручную, что, в свою очередь, позволяет сократить численность команды разработчиков и время, необходимые для осуществления проекта. Если расширить данный пример, включив в него коммерческую систему управления базой данных (СУБД), коммерческий построитель GUI и коммерческое промежуточное ПО, это позволит уменьшить эффективный объем разработки до следующего значения:

> 75 000 строк на языке Ada 95 или C++ при использовании нескольких коммерческих компонентов

Поскольку различие между большим и маленьким проектами оказывает нелинейное воздействие на стоимость жизненного цикла, использование языков самого высокого уровня и соответствующих коммерческих компонентов может существенно повлиять на стоимость. Более того, нередко «проще» является синонимом «лучше»: уменьшение размера обычно делает код более понятным, повышает простоту внесения изменений и надежность. Одним из типичных негативных побочных эффектов является *тенденция к снижению быстродействия при использовании* технологий с более высоким уровнем абстракции, причем одновременно увеличиваются потребляемые ресурсы: процессора, памяти и пропускной способности. Большинство этих недостатков преодолевается за счет улучшения работы аппаратуры и оптимизации. Хотя для встроенных систем такие улучшения оказываются менее эффективными.

3.1.2 Объектно-ориентированные методы и визуальное моделирование

Как известно, в 90-е гг. имело место широкое движение за использование объектно-ориентированной технологии. Я уделяю этой теме мало внимания, поскольку данная технология не имеет отношения к обсуждаемым здесь темам, касающимся управления созданием ПО, и по объектно-ориентированной технологии издано множество книг. В результате некоторых исследований можно прийти к выводу, что объектно-ориентированные языки программирования позволили получить выигрыш как в производительности при создании ПО, так и в качестве ПО [Jones, 1994], хотя экономический выигрыш еще требуется доказать, поскольку стоимость обучения объектно-ориентированным методам разработки, таким как UML, непомерно высока.

Предоставляя более формализованные нотации для отображения и визуализации абстракций ПО, объектно-ориентированная технология оказывает фундаментальное влияние, которое заключается в уменьшении общего размера того, что требуется разработать. Буч описал три другие причины, которые привели к успешному завершению конкретных объектно-ориентированных проектов [Booch, 1996]. Это интересные примеры взаимозависимости между различными направлениями совершенствования экономики ПО. (Цитаты выделены *курсивом*.)

1. *Объектно-ориентированная модель и ее реализация предполагают наличие общего словаря у конечных пользователей системы и у ее разработчиков, что приводит к пониманию решаемой проблемы и теми, и другими.*

   ▲ Это пример того, как объектно-ориентированная технология позволяет вносить соответствующие улучшения в работу команды и в межличностные взаимоотношения.

2. *Использование растянутой во времени интеграции создает возможности для раннего определения риска и внесения необходимых поправок без дестабилизации процесса разработки.*

   ▲ Этот аспект объектно-ориентированной технологии предоставляет возможность использовать процесс упреждающей разработки архитектуры, при котором интеграция является ранней и растянутой во времени процедурой жизненного цикла.

3. *Объектно-ориентированная архитектура предполагает четкое разделение различных элементов системы, создавая надежную защиту, которая позволяет при внесении изменений в одну часть системы сохранить неизменной архитектуру в целом.*

   ▲ Эта особенность объектно-ориентированной технологии критична по отношению к используемым для реализации объектно-ориентированной архитектуры языкам и среде.

Буч также сформулировал пять характеристик успешного объектно-ориентированного проекта:

1. *Требуется уделять самое пристальное внимание развитию системы, что позволяет получить тщательно продуманный набор минимально необходимых характеристик.*
2. *Наличие культуры, которая ставит во главу угла результат, приветствует взаимодействие, но вместе с тем не боится неудач.*
3. *Эффективное использование объектно-ориентированного моделирования.*
4. *Наличие четких представлений об архитектуре.*
5. *Применение грамотно управляемого итерационного и пошагового жизненного цикла разработки.*

Эти характеристики имеют весьма слабое отношение к объектно-ориентированному подходу. Однако объектно-ориентированные методы, нотации и визуальное моделирование обеспечивают мощную технологическую поддержку основы процесса.

3.1.3 Повторное использование

Повторное использование существующих компонентов и создание повторно используемых компонентов являлись естественной деятельностью разработчиков с момента самых первых усовершенствований

языков программирования. Методы разработки ПО всегда неявно имели дело с повторным использованием, которое позволяло минимизировать стоимость работ, не теряя при этом соответствия остальным необходимым параметрам быстродействия, набора функциональных возможностей и качества. Повторное использование не получило в сообществе разработчиков ПО должного признания своей важности лишь потому, что оно не применяется должным образом. В других областях промышленности, связанных с разработкой и производством, повторное использование является более или менее основополагающим приемом, а не каким-нибудь вынужденным технологическим прорывом. Я пытаюсь рассматривать повторное использование как вполне обыденную составляющую достижения возврата инвестиций. Обычная архитектура, обычные процессы, предшествующий опыт и обычная среда — это все примеры повторного использования.

Одним из наиболее сложных препятствий на пути повторного использования всегда была фрагментарность языков, операционных систем, нотаций, архитектуры компьютеров, инструментария и даже «стандартов». В качестве контрпримера можно привести высочайший уровень повторного использования, который стал возможен благодаря успеху компании Microsoft на платформе персональных компьютеров.

Вообще говоря, вещи используются повторно из экономических соображений. Значит, ключевой способ определения того, действительно ли компонент (класс компонентов, коммерческий продукт) допускает повторное использование, заключается в выяснении, не делает ли кто-нибудь на нем деньги. Повторное использование компонентов без этого экономического мотива встречается редко. Остерегайтесь «открытых» библиотек повторного использования, спонсируемых некоммерческими организациями. У них отсутствует экономическая мотивация, им нельзя доверять, там никто не несет ответственности за качество, поддержку, совершенствование и пригодность к применению. Наиболее часто компоненты, представляющие ценность для повторного использования, переделываются в коммерческие продукты, которые поддерживаются организациями, обладающими следующими признаками:

- У них имеется экономическая мотивация продолжительной поддержки.
- Они занимаются улучшением качества продукта, добавляют новые возможности и переходят к новым технологиям.
- У них достаточно широкая клиентская база для того, чтобы получать прибыль.

Определение стоимости разработки компонента повторного использования — нетривиальная задача. На рис. 3.1 рассматриваются накладные расходы. Быстрый рост кривой в начале иллюстрирует экономические препятствия на пути создания компонента повторного использования. Сложно придумать убедительный бизнес-план для разработки, если целью не является поддержка повторного использования во многих проектах. Положительные примеры крайне редки для организаций, занимающихся разработкой ПО, если они не преследуют цели продажи

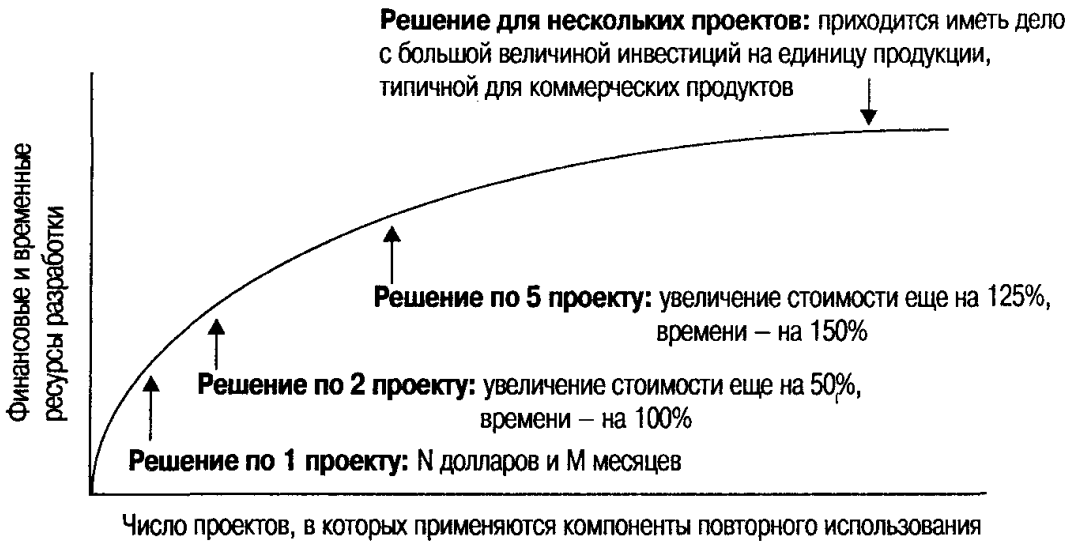


Рис. 3.1. *Вложения и затраты времени, необходимые для получения компонентов повторного использования*

коммерческих компонентов в качестве основной линии своего бизнеса. Большинство компаний не в состоянии экономически конкурировать с устоявшимися коммерческими организациями, чьи инвестиции в большой степени амортизируются за счет клиентской базы. Для достижения успеха на рынке коммерческих компонентов организации требуются три устойчивых элемента: группы разработчиков, инфраструктуры сопровождения и инфраструктуры продаж и маркетинга, ориентированной на продукт. Другим соображением является то, что сложность и стоимость разработки компонентов повторного использования часто недооцениваются.

Выгоды от повторного использования могут оказаться весьма существенными, однако я никогда не являлся сторонником выделения повторного использования в отдельную «технологию». Повторное использование — важная дисциплина, которая оказывает большое влияние на эффективность всех направлений работ и на качество изделий. Я рассматриваю его как синоним возврата инвестиций, который следует принимать во внимание практически во всех видах деятельности и при принятии почти любого решения. Известно немного случаев успешного повторного использования компонентов ПО, не считая коммерческих продуктов, таких как операционные системы, системы управления базами данных, программно-аппаратные комплексы, сети, построители GUI и офисные приложения. С другой стороны, во всех случаях успешного создания ПО, вероятно, применялись некие ключевые аспекты повторного использования (хотя и называвшиеся по-другому) для достижения эффективных результатов.

3.1.4 Коммерческие компоненты

На сегодняшний день распространенным во многих областях подходом является максимальное увеличение интеграции с коммерческими компонентами и готовыми продуктами. Использование коммерческих

компонентов желательно как средство уменьшения объема разработок, выполняемых на заказ, однако на практике все оказывается не так просто. В таблице 3.3 приведены некоторые достоинства и недостатки применения коммерческих компонентов. (Эти соглашения, в частности, особенно актуальны для областей, критичных к выполнению задания.) Поскольку соглашения зачастую оказывают глобальный эффект на качество, стоимость и легкость сопровождения, выбор коммерческих компонентов при разработке, выполняемой на заказ, заметно влияет на общую архитектуру проекта. В этом случае первостепенную важность имеет идея (см. главу 7), что такие решения должны приниматься на ранних стадиях жизненного цикла как составная часть процесса разработки архитектуры.

Таблица 3.3.
Достоинства и недостатки коммерческих компонентов по сравнению с ПО, создаваемым на заказ

Подход	Достоинства	Недостатки
Коммерческие компоненты	Заранее известная стоимость лицензии Широко используемая, устоявшаяся технология Непосредственная доступность Наличие организации, осуществляющей поддержку Аппаратная/программная независимость Разнообразные функциональные возможности	Частое появление новых версий Авансовая оплата лицензии Необходимость постоянной оплаты сопровождения Зависимость от поставщика Потеря эффективности при выполнении Функциональные ограничения Интеграция не всегда тривиальна Отсутствие средств контроля за новыми версиями и сопровождением Ненужные возможности, на которые тратятся дополнительные ресурсы Зачастую недостаточная надежность и стабильность Несовместимость у различных поставщиков
Разработка на заказ	Полная свобода внесения изменений Меньшая по объему, более простая реализация Зачастую большее быстродействие Контроль за разработкой и развитием	Дорогая непредсказуемая разработка Непредсказуемый срок создания Неопределенная модель сопровождения Зачастую незрелый и подверженный дефектам продукт Зависимость от одной платформы Отвлекает внимание экспертов

3.2 СОВЕРШЕНСТВОВАНИЕ ПРОЦЕССА СОЗДАНИЯ ПО

Термин «процесс» слишком перегружен. В организациях, ориентированных на создание ПО, существует много процессов разного масштаба. Я буду использовать слово «процесс» в трех различных значениях.

- *Метапроцесс:* политика, приемы и практика, которые присущи некоей организации при ведении интенсивного бизнеса, связанного с ПО. В центре этого процесса находятся экономика организации, долговременная стратегия и возврат инвестиций в ПО.
- *Макропроцесс:* политика, приемы и практика, которые присущи некоему проекту по созданию законченного ПО с учетом определенных ограничений по стоимости, срокам и качеству. В центре этого процесса находится создание адекватного варианта метапроцесса для конкретного набора ограничений.
- *Микропроцесс:* политика, приемы и практика, присущие команде разработчиков некоего проекта и направленные на получение некоторых результатов в процессе создания ПО. Главным для микропроцесса является создание промежуточного продукта адекватного качества с адекватными функциональными возможностями настолько экономично и быстро, насколько это осуществимо на практике.

Эти три уровня процесса несколько перекрывают друг друга, но у них разные цели, участники, метрики, отношения и временной масштаб (см. таблицу 3.4). Макропроцесс — это процесс уровня проекта в целом, который оказывает влияние на модель оценки стоимости, обсуждаемую в данной главе.

Для достижения успеха большинству проектов по созданию ПО требуется невероятно сложная паутина последовательных и параллельных шагов. По мере роста проекта приходится включать дополнительные шаги лишь для того, чтобы управиться с этой сложной паутиной. Все процессы, связанные с проектом, состоят из производительных и непроизводительных видов деятельности. Производительные виды деятельности приводят к осязаемому прогрессу на пути к конечному продукту. Если речь идет о создании ПО, то в эти виды деятельности входят создание прототипа, моделирование, кодирование, отладка и подготовка документации для пользователей. Непроизводительные виды деятельности, которые не имеют непосредственного влияния на конечный продукт, включают в себя подготовку планов и документации, мониторинг хода работ, оценку рисков, финансовые оценки, управление конфигурацией, оценку качества, интеграцию, тестирование, отбраковку и доработку на поздних стадиях, управление, подготовку персонала, бизнес-администрирование, а также решение ряда других задач. К непроизводительным видам деятельности относятся такие работы, которые повышают конечную ценность продукта, но в общем случае, чем меньше сил на них тратится, тем больше усилий может быть направлено на производительные виды деятельности. Целью улучшения процесса является максимизация ресурсов,

направленных на производительные виды деятельности, и минимизация влияния непроизводительных видов деятельности на такие ресурсы, как персонал, компьютеры и график.

Таблица 3.4.
Три уровня процесса и их атрибуты

Атрибуты	Метапроцесс	Макропроцесс	Микропроцесс
Предмет	Стратегия бизнеса	Проект	Итерация
Цели	Прибыльность стратегии бизнеса Конкурентоспособность	Прибыльность проекта Управление рисками Бюджет проекта, сроки, качество	Управление ресурсами Разрешение рисков Промежуточный бюджет, сроки, качество
Участники	Лица, ответственные за приобретение, заказчики Менеджеры организации	Менеджеры проекта Разработчики ПО	Менеджеры составляющих проектов Разработчики ПО
Метрики	Предсказуемость проекта Получение дохода, контролируемый сегмент рынка	Необходимость уложиться в бюджет и в сроки Успешное достижение основных контрольных точек Отбраковка и переделка в рамках проекта	Необходимость уложиться в бюджет и в сроки Успешное выполнение основных контрольных точек Отбраковка и переделка в рамках отдельных итераций/версий
Отношения	Бюрократия против стандартизации	Качество против финансовой эффективности	Содержание против сроков
Временной масштаб	От 6 до 12 месяцев	От 1 года до нескольких лет	От 1 до 6 месяцев

Возможно, кого-то раздражает то, что я отношу отбраковку и переделку на поздних стадиях, а также подготовку персонала к категории непроизводительных видов деятельности, которые следует минимизировать. Я специально определил отбраковку и переделку как *позднюю*, чтобы отличать их от отбраковки и переделки, являющихся естественным побочным продуктом при создании прототипа. Отбраковка и переделка на ранних стадиях являются производственной необходимостью для большинства проектов, так как позволяют определить бесчисленное множество неизвестных в области принятия решений. Но они крайне нежелательны на поздних стадиях жизненного цикла. Совершенно очевидно, что для хорошего процесса поздние отбраковка и переделка абсолютно излишни.

Многие будут спорить, доказывая, что подготовка персонала не может быть плохой вещью, но мы утверждаем это с точки зрения проекта.

Подготовка персонала является организационной задачей, а не задачей в рамках выполнения проекта. Любому менеджеру проекта, которому приходится нести бремя обучения людей процессам, технологиям или инструментарию, приходится намного хуже, чем менеджеру проекта, который располагает полностью подготовленной рабочей силой. Возможно, обеспечение всех проектов подготовленными работниками не представляется возможным, но при прочих равных условиях привлечение подготовленных людей всегда лучше, чем неподготовленных. В этом смысле подготовка не может рассматриваться как вид деятельности, повышающей конечную ценность продукта.

Качество процесса создания ПО сильно влияет на объем работы, который необходимо выполнить, и, следовательно, на сроки создания программного продукта. На практике отличие хорошего процесса от плохого способно изменять общую оценку стоимости на величину от 50% до 100%, а уменьшение объема работ позволяет сократить сроки выполнения. Лучший процесс может оказать еще более существенное влияние на сокращение времени, требующегося команде для получения представления о продукте известного качества. Почему это так?

Сокращение сроков возможно по крайней мере по трем направлениям.

1. Рассматривая процесс, состоящий из N шагов, мы можем повышать эффективность каждого шага.
2. Рассматривая процесс, состоящий из N шагов, мы можем исключить некоторые шаги, получив процесс, состоящий из M шагов.
3. Рассматривая процесс, состоящий из N шагов, мы можем использовать более конкурентоспособные виды деятельности или ресурсы.

Многие организационные стратегии, ускоряющие выход продукта на рынок, делают упор на первом направлении. Однако в центре внимания большинства усовершенствований процесса, описываемых в настоящей книге, лежит стремление к достижению результатов по второму и третьему направлениям, в которых сокрыт гораздо больший потенциал. В частности, совершенствуя процесс, необходимо уделять внимание нахождению адекватного решения за минимальное количество итераций и исключению, насколько это возможно, отбраковки и переделки, требующих возврата к более ранним стадиям.

Каждый случай переделки предполагает повторное решение соответствующего множества задач. Представим себе, что команда, завершая последовательные действия по анализу, разработке, кодированию и тестированию некоей функциональной возможности, при тестировании обнаруживает изъян. В этом случае приходится последовательно выполнять процедуры повторной разработки, повторного кодирования и повторного тестирования. Последовательное выполнение задач является самым главным препятствием на пути к сокращению сроков. Несмотря на некоторые технологические приемы, позволяющие исключать из процесса целые шаги, самой важной задачей улучшения процесса является сокращение отбраковки и переделки на поздних стадиях жизненного цикла.

В идеальном мире разработки ПО с безупречно сформулированной проблемой, очевидным пространством решений, командой разработчиков, состоящей из опытных гениев, с необходимыми ресурсами и заинтересованными сторонами, объединенными общими целями, мы могли бы завершить процесс разработки ПО за одну итерацию практически без всяких отбраковок и переделок. Однако мир не идеален, и нам следует так управлять процессом разработки, чтобы вид кривой отбраковок и переделок не влиял на условия достижения успеха любой из заинтересованных сторон. Это утверждение служит основополагающей предпосылкой практически для всех улучшений процесса.

3.3 УВЕЛИЧЕНИЕ ЭФФЕКТИВНОСТИ РАБОТЫ КОМАНДЫ

Давно известно, что различия в квалификации персонала могут оказывать большое влияние на продуктивность. Исходная модель COCOMO, например, предполагает, что совместный эффект навыков и опыта персонала может изменять производительность в четыре раза. Это разница между командой неопытных любителей и командой маститых экспертов. На практике представляется рискованным оценивать некую конкретную команду как выходящую за рамки этой шкалы в ту или другую сторону. Большая команда, состоящая, скажем, из 50 человек, всегда комплектуется обычными людьми с обычным опытом. Невозможно обеспечить некий нетривиальный проект сотрудниками, каждый из которых обладал бы оптимальным опытом, был бы полностью подготовлен для использования инструментария и технологий и имел бы коэффициент интеллекта, больший 130. Если даже это удастся провести в жизнь, то команда, скорее всего, окажется недееспособной. Поэтому старый лозунг «Просто нанимайте хороших людей» следует применять с осторожностью. Правильнее будет сказать «Просто собирайте хорошую команду».

Равновесие и охват являются двумя наиболее важными аспектами любой отличной команды. Если в команде отсутствует равновесие, она оказывается уязвимой. По спортивной аналогии, команда разработчиков ПО напоминает футбольную команду (Здесь и далее, когда автор говорит о футболе, он имеет в виду футбол американский. – *Прим. пер.*), в которой требуется наличие различных навыков. Редко можно встретить великую футбольную команду, не имеющую широкого охвата: нападающих, защитников и специальных команд – тренеров и обслуживающего персонала, основных и резервных игроков, пасующих и бегущих. Великим командам необходимо наличие на всех ключевых позициях индивидуально сильных игроков. Команда, перегруженная суперзвездами, каждая из которых стремится к установлению личных рекордов и борется за право считаться лидером, может быть остановлена сбалансированной командой хороших игроков, обладающей несколькими лидерами, которые нацелены на достижение командного результата – победы в матче.

Работа в команде представляется более важной, чем сумма индивидуальностей. Управляя командами разработчиков ПО, менеджеру проекта

необходимо соблюдать равновесие между очень талантливыми и просто высококвалифицированными людьми на ключевых позициях. Ниже приводятся некоторые принципы управления командой:

- Хорошо управляемый проект может быть успешно выполнен обычной командой разработчиков.
- Плохо управляемый проект почти никогда не будет успешным, даже если над ним трудится группа разработчиков-экспертов.
- Система, архитектура которой разработана правильно, может быть реализована и обычной командой создателей ПО.
- Система с плохой архитектурой заставит путаться даже команду разработчиков-экспертов.

Изучая проблемы найма исполнителей для проектов по созданию ПО, Боэм предложил следующие пять принципов [Boehm, 1981]. (Цитаты выделены *курсивом.*)

1. *Принцип верховенства таланта: используйте меньшее количество лучших специалистов.*

   ▲ Этот принцип фундаментален, но применять его можно до определенного предела. Для большинства работ существует «естественный» размер команды, и отклонения в ту или другую сторону нарушают командную динамику, поскольку приводят либо к слишком малому, либо к чересчур большому давлению на отдельных работников.

2. *Принцип соответствия заданий: подбирайте задачи в соответствии с навыками и мотивацией специалистов, имеющихся в наличии.*

   ▲ Этот принцип кажется очевидным. В футбольной команде следует использовать лидера в качестве тренера, хорошего пасующего — в качестве квотербека, быстрого бегуна — в качестве принимающего, а 300-фунтового задиру — в качестве линейного. В случае с разработчиками ПО гораздо сложнее определить наиболее значимые навыки персонала и оптимально распределить задания. Персональные устремления также усложняют эту задачу. В футболе 300-фунтовый линейный не будет даже мечтать о том, чтобы его перевели в квотербеки; слишком уж очевидно различие в необходимом наборе навыков. Однако в командах разработчиков ПО довольно распространено явление, когда талантливые программисты пытаются заниматься разработкой архитектуры или менеджментом. Мне кажется, что в обоих случаях наборы требуемых навыков совершенно различны, поскольку большинство блестящих программистов по природе своей не способны разрабатывать архитектуру или быть менеджерами, и наоборот. Несмотря на это, отдельные лица и даже организации, в которых они работают, рассматривают такие продвижения как желательные. Известно бесконечное число примеров, когда великолепные разработчики ПО непродуманно выдвигались на должности, для работы на которых у них не было ни навыков, ни квалификации. Таким образом происходит замена игрока А на игрока В, в результате которой игрок А переходит с умеренной по влиятельности позиции на высокую, а игрок В — на еще более влиятельную. Двойная ошибка.

3. *Принцип карьерного роста: лучшее, что может сделать организация для своих сотрудников в долговременном плане, помочь им найти себя.*

   ▲ Хорошие работники обычно находят себя в любой среде. Организация в состоянии как помочь, так и помешать сотруднику в поисках своего места, но усилия организации могут принести более всего пользы сотрудникам средним и ниже среднего. Программы подготовки, предлагаемые организациями, обычно являются стратегическими мероприятиями, обладающими образовательной ценностью. Программы подготовки, предусмотренные в рамках проектов, чисто тактические и рассчитаны на то, что они окажутся полезными и будут использоваться на следующий день после окончания подготовки.

4. *Принцип командного равновесия: подбирайте людей, которые дополняют друг друга и гармонично сочетаются друг с другом.*

   ▲ Этот принцип звучит немного сентиментально, но его дух является самым главным фактором для хорошей командной работы. Равновесие команды разработчиков имеет много измерений, и если оно оказывается нарушенным хотя бы в одном из них, выполнение проекта оказывается под серьезной угрозой. Эти измерения включают в себя:

   Исходные качества: интеллект, целеустремленность, творческий подход, организованность, аналитическое мышление

   Психологическая составляющая: лидеры и последователи, сторонники риска и консерваторы, мечтатели и придиры, циники и оптимисты

   Цели: финансовые, набор функциональных возможностей, качество, своевременность

5. *Принцип исключения: человек, выпадающий из команды, никому не принесет пользы.*

   ▲ Этот принцип является следствием предыдущих четырех. Наличие неудачника — это повод подыскать более подходящего работника или остаться с меньшим количеством людей. Неудачник лишает мотивации других членов команды, не может обрести себя и нарушает равновесие в том или ином измерении. Неудачников легко отличить, и почти никогда не следует затягивать с их исключением из команды.

Разработка ПО – командный вид спорта. Менеджеры должны лелеять культуру работы в команде, и командные результаты имеют большее значение, чем индивидуальные достижения. Из этих пяти принципов главными целями следует признать достижение равновесия в команде и соответствие заданий. Принцип верховенства таланта и принцип исключения из команды являются вторичными целями, поскольку их применяют для достижения равновесия в команде. Что же касается карьерного роста, то этой практики работы с кадрами следует придерживаться. Однако те сотрудники или организации, которые уделяют ей внимание в ущерб успеху всей команды, долго на рынке не протянут.

Менеджеры проектов должны обладать многими качествами лидера для того, чтобы повышать эффективность команды. Это имеет значение в любой сфере деятельности. Ниже приводятся те качества менеджеров проектов, которые заслуживают пристального внимания:

1. Умение нанимать. Немногие решения важны так же, как решения о найме. Необходимость правильно подыскать работу для каждого человека кажется очевидной, но является трудноразрешимой задачей.
2. Умение взаимодействовать с заказчиком. Умение избегать враждебности во взаимоотношениях между заинтересованными сторонами является предпосылкой успеха.
3. Умение принимать решения. В бесконечно большом числе книг по менеджменту так и не было дано ясного определения этому атрибуту. В процессе работы мы все понимаем, что это хороший лидер, а умение принимать решения кажется очевидным, несмотря на нечеткое определение.
4. Умение создавать команду. Работа в команде требует от менеджера умения устанавливать доверительные отношения, стимулировать прогресс, заставлять трудиться эксцентричных «примадонн», превращать средних людей в исполнителей высшего класса, избавляться от неудачников и объединять различные мнения в рамках единого направления командной работы.
5. Умение убеждать. Менеджер проекта должен убедить все заинтересованные стороны (включая самого себя) в правильности принятых решений и расставленных приоритетов, убедить кандидатов занять соответствующие должности, убедить в необходимости внесения изменений в статус-кво в случае возникновения препятствий, доказать, что достижения соответствуют поставленным целям. На практике убеждение требует продолжительных переговоров, компромиссов и сопереживания.

3.4 ПОВЫШЕНИЕ УРОВНЯ АВТОМАТИЗАЦИИ ЗА СЧЕТ СРЕДЫ РАЗРАБОТКИ

Инструментарий и среда, используемые при создании ПО, в общем имеют линейное влияние на производительность процесса. Инструментарий для планирования, управления требованиями, визуального моделирования, компиляторы, редакторы, отладчики, инструментарий для проверки соответствия качеству, тестирования и пользовательские интерфейсы обеспечивают необходимую автоматизацию получения результатов в процессе создания ПО. Кроме того, среда управления конфигурацией является основой для выполнения и инструментального оснащения процесса. В первом приближении, чистое влияние инструментария и автоматизации позволяет сократить объем работ на величину от 20% до 40%. Однако, поскольку инструментарий и среду следует рассматривать как основное средство обеспечения автоматизации и улучшения процесса, их влияние может оказаться гораздо большим.

Раздел 3.2 посвящен усовершенствованиям процесса, которые позволяют уменьшить количество отбраковок и доработок, исключить некоторые шаги и минимизировать число итераций. Другой формой совершенствования процесса является повышение эффективности конкретных шагов. Здесь свой важнейший вклад должна внести среда, а именно: требуется автоматизировать задачи, выполнение которых вручную неэффективно или чревато ошибками. Переход к более зрелому процессу создания ПО приводит к появлению новых решений и возможностей в области контроля, осуществляемого менеджерами, над параллельными видами деятельности для достижения осязаемого прогресса и для оценки качества. Опыт выполнения проектов показал, что интегрированная в высокой степени среда является необходимой как для упрощения, так и для усиления контроля за процессом, осуществляемого менеджерами. Среда, обеспечивающая семантическую интеграцию (при которой среде понятен в деталях смысл результатов разработки) и автоматизацию процесса, может способствовать росту производительности, повышению качества ПО и ускорению процесса освоения современных методов. Среда, которая поддерживает пошаговую компиляцию, автоматизированное построение системы и интегрированное регрессионное тестирование, может обеспечить ускорение цикла при итерационной разработке и позволяет командам разработчиков выполнять итерации более свободно.

Серьезное внимание при современном подходе уделяется определению среды разработки и сопровождения как предмету первой необходимости для процесса разработки. Стабильная интегрированная среда разработки обязана поддерживать автоматизацию процесса разработки. Такая среда должна включать в себя управление требованиями, автоматизацию документирования, исходные/целевые (host/target) средства программирования, автоматизированное регрессионное тестирование, постоянное и интегрированное управление изменениями и отслеживание дефектов. Основная линия для успешного хода проекта заключается в найме хороших исполнителей и в обеспечении их хорошими инструментами для выполнения своей работы. Автоматизация процесса разработки позволяет получить выигрыш в качестве, в возможности оценки стоимости и сроков и в общей продуктивности при использовании меньшей команды. Интегрированные наборы инструментов играют все более важную роль в пошаговой/итерационной разработке, позволяя разработчикам быстро ориентироваться в получаемых продуктах и своевременно вносить в них изменения.

Разработка по «круговому» (round-trip) принципу — это термин, используемый для описания ключевых возможностей среды, которая поддерживает итерационный подход. По мере того как различные продукты разработки оказываются размещенными в различных хранилищах информации, возникает необходимость в гарантии эффективного и безошибочного переноса данных из одних продуктов в другие. *Прямое проектирование (forward engineering)* — это автоматизация процесса получения некоторого продукта из более абстрактного представления. Например, компиляторы и загрузчики обеспечивают автоматическое преобразование исходного кода в выполняемый код. *Обратное проектирование (reverse*

engineering) — это генерация или модификация более абстрактного представления из существующего продукта (например, создание визуальной модели из исходного кода).

«Круговая» разработка описывает такую поддержку среды, которая позволяет свободно вносить изменения в отдельные продукты. При этом остальные продукты автоматически изменяются таким образом, чтобы весь набор продуктов, связанных с требованиями, разработкой, реализацией и вводом в действие, сохранял свою непротиворечивость (см. главу 12).

По мере того как в архитектуре стали использоваться разнородные компоненты, платформы и языки, сложность создания, управления и сопровождения крупномасштабных сетей компонентов привела к появлению новых потребностей в области управления конфигурацией и автоматизации управления процессом создания. Однако степень поддержки автоматизации существующих на сегодняшний день сред далека от желаемой. Например, автоматизированное построение тестового варианта из описаний варианта использования и сценария до сих пор находится на такой стадии развития, при которой не поддерживается ничего, кроме наиболее тривиальных случаев, таких как сценарии для тестирования отдельных элементов.

Описывая экономические выгоды, связанные с инструментарием и различными средами, следует сделать одно предостережение. Зачастую поставщики инструментов дают относительно точные индивидуальные оценки потенциального экономического эффекта от использования их инструментов в течение всего жизненного цикла. Например, в какой-нибудь отдельной «инструментальной» нише часто можно встретить подобные утверждения:

- На анализ требований и деятельность, направленную на развитие, тратится 40% общих затрат в жизненном цикле.
- Деятельность по разработке ПО оказывает влияние на более чем 50% ресурсов.
- Этапы кодирования и тестирования составляют свыше 50% от всей работы и занимают более 50% всего времени.
- Проведение тестирования может потребовать до 50% всех ресурсов проекта.
- Управление конфигурацией и изменениями — это критичные виды деятельности, на которые может расходоваться до 25% ресурсов крупномасштабного проекта.
- На создание документации может тратиться более 30% ресурсов, отпущенных на разработку проекта.
- Управление проектом, бизнес-администрирование и оценка прогресса могут стоить до 30% от бюджета проекта.

Если рассматривать их по отдельности, то каждое из них нельзя назвать неправильным, они просто слишком примитивны. (Если верить им всем, то оказывается, что на выполнение большинства проектов требуется до 275% времени и денег!) Объединенные вместе, эти утверждения могут ввести в заблуждение. Остерегайтесь заключений такого рода:

> *Данный инструмент для тестирования позволяет увеличить продуктивность тестирования на 20%. Поскольку выполнение тестирования занимает 50% от всего жизненного цикла, то чистый выигрыш в производительности для проекта в целом составит 10%. При общем бюджете в один миллион долларов на инструменты для тестирования можно потратить $100 000.*

Взаимоотношения между всеми видами деятельности по разработке ПО оказываются слишком сложными для того, чтобы такие простые рассуждения могли быть верными. По моему опыту, суммарный эффект от использования всех инструментов обычно ниже 40%, и большая часть этого выигрыша не может быть получена без внесения соответствующих изменений в процесс. Маловероятно, чтобы какой-либо отдельный инструмент мог повысить общую производительность проекта более чем на 5%. Вообще говоря, лучше относить большинство утверждений поставщиков к виртуальным 275%, чем к суммарным 100%, с которыми приходится иметь дело в реальной жизни.

3.5 ДОСТИЖЕНИЕ НЕОБХОДИМОГО КАЧЕСТВА

Многое из того, что рассматривается сегодня в качестве лучших способов создания ПО, является следствием процесса разработки и технологий, о которых идет речь в этой главе. В дополнение ко всему, эти способы оказывают влияние на повышение финансовой эффективности. Многие из них позволяют также улучшать качество без изменения цены. В таблице 3.5 приводятся некоторые аспекты повышения качества.

Ключевыми способами улучшения общего качества ПО являются следующие:

- Уделять особое внимание ведущим требованиям и критичным вариантам использования на ранних стадиях жизненного цикла. Уделять внимание полноте и соответствию всем требованиям на поздних стадиях жизненного цикла. В течение всего жизненного цикла уделять внимание поддержанию равновесия между эволюцией требований и эволюцией разработки, с одной стороны, и эволюцией планов, с другой.
- Использовать метрики и показатели для измерения прогресса и качества архитектуры по мере того, как она развивается от прототипа высокого уровня до полностью готового продукта.

- В течение всего жизненного цикла обеспечивать среду, поддерживающую раннее и постоянное управление конфигурацией и изменениями, строгие методы разработки, автоматизацию документирования и автоматизацию регрессионного тестирования.
- Использовать визуальное моделирование и языки высокого уровня, которые поддерживают управление архитектурой, абстракцию, надежное программирование, повторное использование и самодокументирование.
- Рано и постоянно вникать в проблемы производительности с помощью оценок, основанных на демонстрациях.

Таблица 3.5.
Общее повышение качества в современном процессе

Фактор, влияющий на качество	Традиционный процесс	Современный итерационный процесс
Недостаточное понимание требований	Поздно обнаруживается	Рано выявляется
Риски разработки	Неизвестны до последнего момента	Оказываются понятными и разрешаются на ранних этапах
Коммерческие компоненты	Большей частью недоступны	Непосредственно влияют на качество, однако решения по их использованию должны приниматься на ранних стадиях жизненного цикла
Управление изменениями	На поздних этапах жизненного цикла, хаотичное и приносящее вред	На ранних этапах жизненного цикла, непосредственное и в простой форме
Ошибки разработки	Вскрываются поздно	Разрешаются рано
Автоматизация	Преимущественно чреватые ошибками процедуры, выполняемые вручную	Преимущественно автоматизированное преобразование продуктов с минимальными ошибками
Достаточность ресурсов	Непредсказуемая	Предсказуемая
Сроки	Чрезмерно ограниченные	Определяются качеством, производительностью и технологией
Результирующая производительность	«Бумажный» анализ или раздельная имитация	Выполняемые прототипы, ранняя обратная связь, количественные оценки
Строгость процесса создания ПО	Основан на документации	Управляемый, измеряемый и поддерживаемый инструментами

Дополнительные возможности вникать в проблемы производительности работающей системы становятся все более важными, поскольку проекты включают в себя как коммерческие компоненты, так и компоненты, произведенные на заказ. В традиционном процессе разработки ПО особое внимание уделяется раннему получению оценок размера и времени, используемых компьютерной программой. Однако типичной хронологической последовательностью при оценке производительности является следующая:

- Начало проекта. Предложенный проект утверждается как малорискованный с адекватными границами производительности.
- Начальный обзор разработки. Оптимистические оценки адекватных границ при разработке базируются преимущественно на «бумажном» анализе и грубых моделях критичных вариантов. В большинстве случаев к этому моменту становится абсолютно ясно, какие на самом деле будут использоваться алгоритмы и каковы будут размеры баз данных. Однако полная инфраструктура, включая затраты на операционную систему, систему управления базой данных и на межпроцессное и сетевое взаимодействие, а также все вторичные процессы обычно еще неясны.
- Рассмотрение разработки в середине жизненного цикла. Границы оценок постепенно сужаются по мере того, как первые контрольные цифры и начальные тесты начинают подтверждать тот оптимизм, который был присущ более ранним оценкам.
- Интеграция и тестирование. Обнаруживаются серьезные проблемы с производительностью, возникает необходимость внесения фундаментальных изменений в архитектуру. Обычно во всем винят лежащую в основе инфраструктуру, хотя настоящими виновниками являются неправильное использование инфраструктуры, несовершенные архитектурные решения либо плохо понятые ранние договоренности по разработке.

Такая последовательность имеет место, поскольку первые представления о производительности базировались исключительно на наивных оценках, сделанных разработчиками на основе бесчисленного множества критериев. В распределенных системах, состоящих из множества взаимодействующих между собой компонентов, подход, основанный на демонстрациях, позволяет значительно точнее оценить проблемы производительности. Ранние демонстрации могут осуществляться на платформе, на которой выполняется разработка, или на платформе, на которой будет производиться эксплуатация, либо на частично сконфигурированных сетях. Как бы то ни было, они могут быть спланированы и направлены на то, чтобы стать плодотворным инженерным экспериментом. Проблемы с производительностью на начальных стадиях типичны. Более того, они могут оказаться полезными, поскольку позволяют выявлять недостатки в архитектуре и слабость коммерческих компонентов на ранних стадиях жизненного цикла, когда еще можно внести необходимые поправки.

3.6 ЭКСПЕРТНЫЕ ОЦЕНКИ: ВЗГЛЯД С ПРАКТИЧЕСКОЙ ТОЧКИ ЗРЕНИЯ

Экспертные оценки зачастую неоправданно рекламируются как ключевой аспект качества системы. Мой опыт говорит о том, что экспертные оценки хороши как второстепенные механизмы. Они редко вносят значительный вклад в качество по сравнению с первостепенными механизмами и показателями качества, на которые следует обращать особое внимание в процессе управления:

- Перевод проектной информации из одной группы продуктов в другую, что позволяет оценивать непротиворечивость, реализуемость, понятность и технологические ограничения, присущие этим рабочим продуктам.
- Демонстрации по достижении основных контрольных точек, которые заставляют оценивать результаты по значимым критериям в контексте конкретных вариантов использования.
- Инструментальная среда (компиляторы, отладчики, анализаторы, автоматизированные системы тестирования), которые гарантируют строгость, непротиворечивость, завершенность представления и контроль изменений.
- Тестирование на протяжении всего жизненного цикла, позволяющее выявлять критичные соотношения ресурсов, критерии приемки и соответствие требованиям.
- Метрики управления изменениями для объективного понимания тенденций изменения с различных точек зрения и степени достижения качества и прогресса разработки.

Я убежден в том, что значение этих оценок сильно преувеличивается, но в определенных случаях от них можно получить значительную отдачу. Одним из ценных качеств проверок является профессиональный рост команды. Вообще говоря, полезно, когда продукт, созданный младшими членами команды, оценивается старшими наставниками. Передача продукта любителей экспертам и наоборот — это хороший механизм ускорения обретения знаний и навыков новым персоналом. Таким образом можно выявить грубые ошибки и установить необходимую обратную связь с тем, чтобы впредь избегать порочной практики. Это один из лучших способов обучения начинающих разработчиков ПО.

Оценки являются также замечательным средством сохранения ответственности авторов за качество их продукта. Тщательное изучение программного продукта и документации всех авторов следует рассматривать как процесс, сопутствующий основному. Лучше, если объектами таких оценок окажутся все авторы, а не все компоненты. Для молодых сотрудников необходимо периодически проводить выборочные оценки, сами же они могут обучаться, оценивая более опытных работников. Варьирование уровня неформальных оценок производится постоянно, когда разработчики просматривают или интегрируют свое ПО с ПО других

авторов, а также при проведении тестирования независимыми группами тестирующих. Однако такая «оценка» в большей степени направлена на обобщенные аспекты и аспекты работы системы в целом.

Наконец, критичный компонент заслуживает того, чтобы его проверили несколько человек, предпочтительно таких, у которых имеется заинтересованность в его качестве, быстродействии или функциональных возможностях. Оценка, направленная на решение некоей существующей проблемы, может оказаться эффективным способом обнаружения ее причины или нахождения метода ее решения.

Несмотря на эти положительные моменты оценок, многие организации преувеличивают значение совещаний и формальных оценок, настаивая на оценках всех разработанных продуктов. Такой подход может оказаться чрезвычайно непроизводительным. Примерно лишь для 20% технических продуктов (таких, как отдельные варианты использования, проектные модели, исходный код и тестовые варианты) детальное рассмотрение оказывается предпочтительным по сравнению с другими, более полезными с точки зрения гарантии качества способами. Процесс, качество которого изначально предполагается гарантировать посредством оценок, не может быть финансово оправдан. В некоторых публикациях особо отмечается важность и высокая степень ROI таких оценок. Я подозреваю, что эти исследования публиковались профессионалами в области контроля качества, которые склонны преувеличивать важность своей дисциплины. Зачастую мой голос по данной теме остается в одиночестве, но я готов обосновать это логически.

Существенные ошибки разработки или проблемы, касающиеся архитектуры, редко можно обнаружить посредством поверхностного просмотра, если оценка не является узконаправленной на конкретную проблему. А большинство оценок поверхностно. Современные системы чрезвычайно сложны, им присущи наличие неисчислимого множества компонентов, параллельного выполнения, распределенных ресурсов и другие не менее важные аспекты сложности. Для того чтобы охватить динамические взаимодействия даже в рамках простых программных систем для простых случаев, потребовался бы человеческий интеллект сродни интеллекту шахматистов мирового уровня. Следовательно, выборочные оценки человеком имеют тенденцию вырождаться до уровня замечаний по стилю и по семантическим ошибкам первого порядка. Они редко приводят к обнаружению реальных «узких мест», серьезных проблем в управлении (таких, как тупиковые ситуации, конфликты или борьба за ресурсы) или недостатков архитектуры (например, изъянов в масштабируемости, в надежности или в возможностях взаимодействия). Во всех случаях, кроме самых тривиальных, проблемы с архитектурой выявляются только в результате проведения более строгих технических мероприятий, а именно:

- Анализа, создания прототипов или проведения экспериментов
- Создания проектных моделей
- Перевода текущего состояния модели в выполняемый код

- Демонстрации сильных и слабых сторон текущего состояния разработки в контексте наиболее критичных вариантов использования и сценариев
- Учета уроков в моделях, вариантах использования, при реализации и планировании

Достижение качественной архитектуры присуще итерационному процессу, который сбалансирован по различным группам продуктов. По ходу работы может существовать сколько угодно контрольных точек, в том числе оценки и просмотры критичных моментов человеком. Но эти оценки не являются главными. Продукты более ранних этапов жизненного цикла, конечно же, сильнее зависят от субъективного человеческого взгляда, чем продукты более поздних этапов. Использование оценок человека для большого процента ресурсов проекта является плохой практикой и лишь продлевает существование приносящих мало пользы «оценщиков», которые не оказывают почти никакого влияния на успех процесса. Возьмите любой успешный проект по созданию ПО и спросите у любого ключевого проектировщика, тестировщика или разработчика о том, что явилось определяющим для них в достижении успеха. Маловероятно, что кто-нибудь из них упомянет совещания, оценки или документацию.

Высокое качество — это ответственность каждого, и она должна являться составной частью практически любого вида деятельности внутри процесса, а не быть особым видом деятельности, осуществляемой специалистами по качеству. Оценка и определение качества по основным направлениям разработки должны быть задачей команды разработчиков, независимой от команды архитекторов и команды программистов. Выполняемая ими на протяжении всего жизненного цикла оценка рабочих продуктов обычно включает в себя контроль за изменениями, анализ тенденций и тестирование, в том числе оценки.

Глава 4

Два пути: старый и новый

Ключевые моменты

- Традиционная разработка ПО основывается на многочисленных хорошо устоявшихся принципах. Некоторые из них по-прежнему действенны, другие же устарели.
- Современный процесс управления созданием ПО включает в себя многие из традиционных принципов, но обращается также к существенно обновленным подходам.

За последние два десятилетия в процессе создания ПО произошли значительные изменения. Многие из традиционных управленческих и технических приемов были заменены новыми подходами, которые сочетают в себе опыт успешных проектов и преимущества технологии разработки ПО. Этот переход мотивировался неутолимой жаждой создания ПО с меньшими затратами, большим числом возможностей и за более короткое время в условиях жесткой конкуренции. Наличие в индустрии коммерческого ПО сочетания конкурентного давления, прибыльности, разнородности заказчиков и быстро изменяющейся технологии привело к использованию многими организациями новых подходов к управлению. В оборонной и аэрокосмической промышленности для многих систем требуется новая парадигма управления, отвечающая бюджетным ограничениям, среде с постоянно меняющимися и разнообразными угрозами, длительным срокам эксплуатации систем и превалированию широкомасштабных, сложных приложений.

4.1 ПРИНЦИПЫ ТРАДИЦИОННОЙ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Существует множество описаний «старого пути» в программной инженерии. За годы практической разработки индустрия ПО извлекла большое число уроков и сформулировала множество принципов. В данном разделе описывается один из взглядов на сегодняшние принципы программной инженерии как отправная точка для представления основных тем,

обсуждаемых в остальной части книги. Выбранная мною точка отсчета — это короткая статья «Fifteen principles of software engineering» («Пятнадцать принципов программной инженерии») [Davis, 1994]. Она постепенно переросла в книгу [Davis, 1995], которая содержит в себе 201 принцип. Несмотря на свое название, данная статья описывает тридцать самых важных принципов, и она не хуже всех прочих изложений мудрости традиционного процесса в индустрии ПО. Я готов подписаться под большей частью этих принципов, но я убежден, что некоторые из них устарели. 30 принципов Дэвиса, приводимые ниже, выделяются *курсивом*. Для каждого принципа дается комментарий по поводу того, будет ли этот принцип поддерживаться или изменяться в данной книге. Я сделаю несколько допущений, которые будут оставлены без доказательств до следующих глав.

1. ***Достигайте наивысшего качества.*** *Качество должно иметь количественное выражение, и необходимо задействовать механизмы, стимулирующие его достижение.*

   ▲ Непосредственное описание качества, соответствующего данному проекту, является важным, но это не просто сделать в самом начале проекта. Именно поэтому современный подход к процессу борется за то, чтобы определить компромиссы между возможностями, качеством, стоимостью и сроками на более ранних этапах жизненного цикла. Пока такое понимание отсутствует, определить качество или управлять достижением качества невозможно.

2. ***Создание высококачественного ПО возможно.*** *Методы, повышающие качество, включают в себя привлечение заказчика, создание прототипов, упрощение разработки, проведение проверок и найм лучших сотрудников.*

   ▲ Этот принцип — самый ненужный.

3. ***Передавайте продукты заказчику как можно раньше.*** *Независимо от того, сколь много усилий вы прикладываете для изучения нужд заказчика на этапе описания требований, наиболее эффективный способ определить настоящие нужды – это отдать продукт пользователям, чтобы они его пощупали.*

   ▲ Это ключевой принцип схемы современного процесса, причем в течение жизненного цикла должно существовать несколько различных механизмов привлечения заказчика. В зависимости от области такие механизмы могут включать в себя демонстрационные прототипы, основанные на демонстрациях контрольные точки и альфа/бета-версии.

4. ***Определите проблему до того, как записывать требования.*** *Встречаясь с тем, что кажется проблемой, многие разработчики тут же бросаются предлагать решение. Прежде чем пытаться решить проблему, убедитесь, что изучены все альтернативные возможности. Очевидное решение не должно ослепить вас.*

▲ Этот принцип является прямым указанием на проблемы, присущие традиционному процессу определения требований. Параметры проблемы становятся более понятными по мере нахождения решения. Современный подход к процессу рассматривает проблему совместно с ее решением до тех пор, пока не удастся понять проблему достаточно хорошо для того, чтобы полностью переключиться на производство.

5. ***Оценивайте альтернативы при разработке.*** *После согласования требований необходимо исследовать различные архитектуры и алгоритмы. Вряд ли вы захотите работать с «архитектурой» только потому, что она была использована при определении требований.*

▲ Этот принцип, вероятно, укоренился в «водопадном» сознании в двух видах. (1) Требования предшествуют архитектуре, вместо того чтобы рассматриваться совместно. (2) Архитектура заложена в спецификациях самих требований. В то время как современный процесс в явном виде способствует проведению анализа альтернатив при разработке, это делается параллельно с определением требований, а различные нотации и рабочие продукты, касающиеся требований и архитектуры, явным образом не связаны между собой.

6. ***Используйте подходящую модель процесса.*** *Для каждого проекта необходимо выбирать такой процесс, который окажется наиболее осмысленным для данного проекта с учетом корпоративной культуры, желания рисковать, области применения, легкости изменения требований и того, насколько правильно поняты требования.*

▲ Действительно, ни один отдельно взятый процесс не является универсальным. Термин «схема процесса» я использую для обозначения изменчивого класса процессов, а не какого-то одного окаменевшего примера. В главе 14 обсуждаются настройка и адаптация процесса под конкретные нужды проекта.

7. ***Используйте различные языки на разных стадиях.*** *Присущая нашей индустрии вечная жажда простых решений сложных проблем подвигла многих на то, чтобы заявить, что лучший метод разработки использует одну и ту же нотацию на протяжении всего жизненного цикла. Стоит ли разработчикам ПО применять язык Ada для описания требований, разработки и кодирования, если этот язык не является оптимальным для данных этапов?*

▲ Это важный принцип. В главе 6 описываются подходящая организация и рекомендуемые языки/нотации для основных продуктов процесса.

8. ***Минимизируйте семантический разрыв.*** *Для того чтобы минимизировать семантический разрыв, структура ПО должна быть настолько близка к структуре предметной области, насколько это возможно.*

▲ Этот принцип являлся основной движущей силой разработки объектно-ориентированных методов, компонентно-ориентированной разработки и визуального моделирования.

9. ***Ставьте методы выше инструментов.*** *Недисциплинированный разработчик ПО, вооруженный инструментом, становится опасным недисциплинированным разработчиком ПО.*

   ▲ Этот принцип верен, но здесь пропущено два важных момента. (1) Дисциплинированный разработчик ПО, вооруженный хорошим инструментарием, обойдет дисциплинированных экспертов ПО, лишенных каких-либо инструментов. (2) Один из лучших способов продвигать, стандартизировать и распространять хорошие методы — это использовать автоматизацию.

10. ***Сначала напишите правильную программу, а затем увеличивайте ее быстродействие.*** *Намного легче увеличить быстродействие правильной программы, чем заставить правильно работать быструю программу. Не отвлекайтесь на оптимизацию при первоначальном кодировании.*

    ▲ Это глубокое утверждение. Некоторые эксперты ПО делают неправильные утверждения следующего характера: «Проблемы работы системы ПО на ранних стадиях — это верный признак рисков в дальнейшем». Каждый успешный нетривиальный проект по созданию ПО из тех, с которыми я знаком, сталкивался с проблемами на ранних стадиях жизненного цикла. Я буду спорить с утверждением о том, что почти каждая незрелая архитектура (особенно в случае широкомасштабного проекта) имеет проблемы с работоспособностью при первых выполняемых итерациях. Обладание чем-то выполняемым (работающим) на ранних стадиях является предпосылкой для достижения баланса в отношении производительности. Понять все это посредством анализа оказывается слишком сложно.

11. ***Проверяйте код.*** *Оценка детальной разработки и кода более хороший способ обнаружения ошибок, чем тестирование.*

    ▲ Значение этого принципа переоценивается для всех, кроме самых простейших, систем ПО. Сегодняшние возможности аппаратуры, языки программирования, автоматизированные среды позволяют эффективно выполнять автоматизированный анализ и тестирование на протяжении всего жизненного цикла. Постоянное и автоматизированное тестирование в течение жизненного цикла — это необходимость для любой современной итерационной разработки. Вообще говоря, беспорядочные проверки (в противоположность проверкам, которые сосредоточиваются на известных проблемах) редко позволяют вскрыть недоработки в архитектуре или глобальные недостатки при разработке. Неверно было бы утверждать, что неэффективными являются все проверки. Когда они используются разумно и направлены на определенную проблему, то оказываются чрезвычайно эффективными для ее решения. Однако этот принцип не должен входить в число первых 15, особенно если учесть, что в индустрии ПО перепроверка является обычной практикой.

12. ***Хорошее управление более важно, чем хорошая технология.*** *Лучшая технология не сможет компенсировать слабое управление, а хорошее управление позволяет получить великолепные результаты даже при наличии недостаточных ресурсов. Хорошее управление заставляет полностью выкладываться, хотя и не существует универсальных «правильных» стилей руководства.*

▲ Моя вера в этот принцип и подвигла меня написать эту книгу. Единственным аргументом, по которому можно спорить, является то, что термин «недостаточные ресурсы» двусмыслен. Отличная хорошо управляемая команда может добиваться великолепных результатов при недостаточных финансах и сроках. С другой стороны, хорошее управление и не очень качественная команда являются взаимоисключающими, поскольку хороший менеджер привлечет, настроит и наймет качественную команду.

13. ***Люди – вот ключ к успеху.*** *Наиважнейшим моментом являются умелые люди, которые обладают необходимыми опытом, талантом и подготовкой. Правильно подобранные работники с неподходящими инструментами, языками и процессом добьются успеха! Неправильно подобранные люди с подходящими инструментами, языками и процессом скорее всего потерпят неудачу.*

▲ Этот принцип расположен в списке слишком далеко от его начала.

14. ***Подражайте с осторожностью.*** *Из того только, что все делают нечто, не следует, что это нечто окажется правильным и для вас. Оно может и быть правильным, но следует тщательно оценить его применимость в вашей среде. Объектно-ориентированная технология, метрики, повторное использование, усовершенствование процесса, CASE, создание прототипов – все это может привести к улучшению качества, снижению цены и удовлетворению потребностей пользователя в большей степени. Потенциал таких приемов зачастую переоценивается, а преимущества никоим образом не являются гарантированными или универсальными.*

▲ Это мудрый совет, особенно для быстрорастущей индустрии, в которой технологические изыски трудно отличить от усовершенствований технологии. Возможность компромиссов, стоимость и сроки не всегда сопутствуют современным технологиям.

15. ***Принимайте на себя ответственность.*** *Когда рушится мост, мы спрашиваем: «Что инженеры сделали неправильно?» Если же выходит из строя ПО, мы редко задаемся подобным вопросом. Факты, однако, таковы, что в любой прикладной дисциплине самые лучшие методы могут быть использованы для создания ужасных решений, а самые устаревшие методы – для получения элегантных результатов.*

▲ Это замечательное следствие пункта 14. Для достижения успеха необходимо нечто большее, чем просто набор хороших методов, инструментария и компонентов. Для этого требуются также хорошие работники, хорошее управление и культура обучения, которая устремлена к дальнейшему прогрессу даже при неизбежных столкновениях с бесконечными промежуточными трудностями.

16. ***Выясните приоритеты заказчика.*** *Возможно, заказчика устроит задержка с реализацией 90% функциональных возможностей, если 10% из них он получит вовремя.*

▲ Понимание приоритетов заказчика важно, но только в сочетании с другими промежуточными приоритетами. «Покупатель всегда прав» — это утверждение, вероятно, привело к бессмысленным тратам денег в большей степени, чем любое другое ошибочное представление. Особенно часто заказчик оказывается неправым при заключении правительственных контрактов, хотя это справедливо и в общем случае (при заключении контракта между заказчиком и системным интегратором).

17. ***Чем больше они получают, тем большего хотят.*** *Чем больше функциональных возможностей (чем выше быстродействие) вы предоставляете пользователю, тем больше функциональных возможностей (более высокое быстродействие) ему требуется.*

▲ Это верно, но предполагает, что вы никогда ничего не захотите показывать пользователю. Следовало бы сказать так: «Чем больше пользователи видят, тем больше они начинают понимать». Не все заказчики на 100% руководствуются жадностью. Они знают, что ресурсы их некоторым образом лимитированы, а у разработчиков существуют свои ограничения. Демонстрация промежуточных результатов — это наглядная деятельность, которая необходима для приведения в соответствие ожиданий заказчика. Если распространять данный принцип на современный процесс, это будет означать, что менеджер проекта по созданию ПО должен обладать объективной информацией, которой он сможет оперировать, оспаривая неизбежные просьбы о внесении изменений и поддерживая баланс между допустимостью, функциональными возможностями и риском.

18. ***Сразу запланируйте переделку.*** *Одним из наиболее важных и критичных факторов успеха является то, насколько продукт в целом оказывается нов. Абсолютно новые приложения, архитектура, интерфейсы или алгоритмы редко работают с первого раза.*

▲ Не следует планировать переделку одной из версий. Вместо этого планируйте преобразование продукта из несовершенного прототипа в хорошо продуманное изделие. Если вам придется выбросить его, что ж, выбрасывайте, однако не надо планировать это с самого начала. Возможно, в прошлом это было мудрым советом для проектов, на 100% состоящих из ПО, разрабатываемого на заказ. Однако в современных программных системах используются многие из уже существующих компонентов (по крайней мере, операционная система, СУБД, GUI, сеть и промежуточное ПО), и часть из того, что создается при первом проходе, может быть сохранена.

19. ***Разрабатывайте так, чтобы можно было вносить изменения.*** *Архитектура, компоненты и методы задания спецификаций, которыми вы пользуетесь, должны позволять внесение изменений.*

▲ Это простое утверждение чрезвычайно сложно выполнить на практике. Основной его смысл в том, что мы должны уметь предсказывать будущее и строить схему таким образом, чтобы в нее можно было вносить изменения, которые еще четко не определены. Тем не менее я поддерживаю этот принцип всем сердцем, поскольку он действительно критичен для достижения

успеха. Трудно предсказать будущее точно, но попытки предположить, какие именно изменения вероятнее всего придется вносить в течение жизненного цикла, являются хорошим упражнением по управлению рисками и темой, постоянно повторяющейся в успешных проектах по созданию ПО.

20. ***Разработка без документации разработкой не является.*** *Часто приходится слышать от разработчиков ПО: «Все, я закончил разработку. Теперь осталось только подготовить документацию».*

▲ Этот принцип тоже присущ подходам прошлого, когда документация была ведущей, причем документация и собственно ПО существовали раздельно. Использование визуального моделирования и языков программирования более высокого порядка обычно приводит к тому, что оказывается непроизводительным иметь отдельные документы с целью описания разработки ПО. Документация самого верхнего уровня архитектуры может быть чрезвычайно полезной, если она написана живым языком и кратко, но основными материалами, используемыми командой разработчиков, являются описание проекта, исходный код и основные тесты. Для более полного использования современных технологических преимуществ я бы модифицировал этот принцип следующим образом: «Большинство рабочих продуктов в проекте должны быть самодокументируемыми». (См. главу 6).

21. ***Используйте инструментарий, но оставайтесь реалистами.*** *Инструментарий для разработки ПО позволяет тому, кто его применяет, работать более эффективно.*

▲ Этот принцип упрощает критичный аспект современной разработки ПО: важность среды разработки. Совершенный процесс должен быть хорошо организован, автоматизирован и обеспечен инструментами. Проекты с итерационной разработкой требуют широкой автоматизации. Недальновидно экономить на создании хорошей среды разработки.

22. ***Избегайте рисков.*** *Многие программисты любят создавать программы с различными трюками – конструкциями, которые работают правильно, но непонятно как. Покажите всему миру, насколько вы ловки, избежав трюков в своей программе.*

▲ Мне кажется, трудно поверить в то, что это является одним из главных принципов. Сложно провести границу между «трюком» и новаторским решением. Я знаю, к чему стремится Дэвис, но мне бы не хотелось вводить в действие принцип, который в качестве побочного эффекта зажимал бы новаторство. Запутанных способов кодирования следует избегать до тех пор, пока не появятся непреодолимые причины их использования. К несчастью, такие причины весьма обычны для нетривиальных проектов.

23. ***Пользуйтесь инкапсуляцией.*** *Сокрытие информации является простым, проверенным средством, позволяющим получить ПО, которое проще тестировать и сопровождать.*

▲ Разработка на основе компонентов, объектно-ориентированная и современная разработка, а также системы программирования усовершенствовали этот принцип, сделав его основным в практике. Инкапсуляция является фундаментальным средством разработчика ПО так же, как математика является фундаментальным средством для физика. Ее следовало бы ввести в качестве предмета отдельного курса на целый семестр в университетах, в которых ведется обучение программной инженерии.

24. ***Используйте связность (coupling) и сцепление (cohesion).*** *Связность и сцепление – лучшие средства измерения присущих ПО сопровождаемости и адаптируемости.*

▲ Этот жизненно важный принцип трудно применим на практике. Связность и сцепление — абстрактные характеристики компонентов, для которых, насколько мне известно, не существует устоявшихся объективных определений. Современные способы определения, относящиеся к сопровождаемости и адаптируемости, построены на измерении количества дефектов и доработок. Компоненты, обладающие внутренним сцеплением и минимальной связностью, легче адаптировать с меньшим количеством дефектов и доработок. Мы можем судить о болезни (слишком большая связность и малое сцепление), только наблюдая и измеряя ее симптомы (дефекты и доработки).

25. ***Используйте меру сложности Маккейба (McCabe).*** *Существует много известных способов для определения сложности, присущей ПО, но ни один из них не является настолько интуитивно понятным и легким в применении, как способ Тома Маккейба.*

▲ Способы измерения сложности важны для определения некоторых критичных компонентов, которым требуется повышенное внимание. Однако в моей практике сложность на самом деле была очевидна; редко приходится сталкиваться с тем, чтобы измерения сложности использовались в практических приложениях для управления проектом или принятия решений. Такие измерения могут быть интересны с академической точки зрения (изучение метапроектов и стратегическое принятие решений), а также могут оказаться полезными при управлении проектом (если оно автоматизировано), но вряд ли они относятся к наиважнейшим принципам.

26. ***Не тестируйте свое собственное ПО.*** *Разработчики никогда не должны становиться главными тестирующими своего ПО.*

▲ С этим принципом часто спорят. С одной стороны, независимая тестирующая команда предлагает независимую точку зрения. С другой стороны, разработчикам ПО необходимо принимать на себя ответственность за качество своей продукции. В главе 11 я ратую за обе точки зрения: разработчикам следует тестировать свое собственное ПО, это должна делать и независимая команда.

27. ***Анализируйте возможные причины ошибок.*** *Намного дешевле уменьшить последствия ошибки, предотвратив ее, чем искать ее и вносить исправления. Единственным способом достижения этого является анализ причин возникновения ошибок по мере их обнаружения.*

▲ При поверхностном взгляде кажется, что это хороший принцип, особенно на стадии конструирования (construction), когда высока вероятность повторения ошибок. Однако в результате анализа ошибок в сложных программных системах оказалось, что одним из критичных источников ошибок являются излишний анализ и излишнее проектирование, выполняемые на бумаге на ранних стадиях проекта. В некоторой степени излишняя работа была направлена на предотвращение ошибок. Это приводило к уменьшению отдачи от вложенных ресурсов по сравнению с тем, что могло бы быть получено при переходе к созданию прототипов и построению системы, при котором ошибки оказываются более очевидными и осязаемыми. Таким образом, я бы разбил данное утверждение на два принципа: (1) не бойтесь делать ошибки на стадии разработки, (2) анализируйте причины ошибок на этапе производства.

28. ***Знайте, что энтропия ПО возрастает.*** *Сложность любой программной системы, претерпевающей множественные изменения, возрастает, и она становится все менее и менее организованной.*

▲ Это еще один пережиток традиционной архитектуры ПО. Почти все программные системы подвергаются множественным изменениям, а признаком непродуманной архитектуры является рост энтропии таким образом, что ею трудно управлять. Энтропия имеет тенденцию возрастать в угрожающих размерах, когда интерфейсы начинают изменяться из тактических соображений. Целостность архитектуры — это понятие прежде всего стратегическое, и следить за ней необходимо с особой тщательностью. Современные инструменты управления изменениями заставляют уважать проект и следовать целостности интерфейса. Качественная архитектура — это такая архитектура, энтропия в которой растет минимально, а изменения могут вноситься со стабильным, предсказуемым результатом. Идеальная архитектура позволила бы вносить изменения без какого бы то ни было роста энтропии.

29. ***Люди и время не взаимозаменяемы.*** *Измерение проекта только в человеко-месяцах имеет мало смысла.*

▲ Это вечный принцип.

30. ***Ожидайте лучшего.*** *Ваши работники будут стараться гораздо больше, если вы ожидаете от них многого.*

▲ Этот принцип применим ко всем дисциплинам, а не только к управлению созданием ПО.

Я использовал некоторые провокационные слова в своих комментариях. Я не ставил перед собой цели подтвердить или опровергнуть принципы Девиса, а лишь хотел продемонстрировать свои пристрастия и спровоцировать на размышления. В то время как примерно половина принципов, на мой взгляд, имеет огромную пользу, вторая половина либо нуждается в изменении приоритетов, либо была вытеснена новой технологией.

4.2 СОВРЕМЕННЫЕ ПРИНЦИПЫ УПРАВЛЕНИЯ СОЗДАНИЕМ ПО

Сегодняшние принципы управления созданием ПО, описанные в разделе 4.1, развиваются и совершенствуются на базе традиционных методов, однако в них не выделены современные принципы, на которых основывается данная книга. Вот мои первые 10 принципов управления созданием ПО, представленные в том же формате, что и принципы Девиса. (Пять первых, являющихся основными для моего определения итерационного процесса, изображены на рис. 4.1.) Все принципы расположены в порядке своей значимости, а слова, выделенные ***жирным курсивом***, используются в дальнейшем в качестве сокращенного варианта развернутых определений.

1. **Основывайте процесс на *упреждающей разработке архитектуры.*** Это потребует достижения очевидного баланса между ведущими требованиями, архитектурно значимыми решениями при разработке и планированием жизненного цикла до того, как будут задействованы ресурсы для полномасштабной разработки.

2. **Стройте *итерационный процесс жизненного цикла* таким образом, чтобы он сталкивался с рисками на ранних этапах.** В сегодняшних сложнейших программных системах совершенно невозможно последовательно описать проблему целиком, полностью разработать ее решение, создать ПО, после чего протестировать конечный продукт. Напротив, итерационный процесс улучшает понимание проблемы, повышает эффективность решения и плана действий и способствует взвешенному рассмотрению всех задач разработчика. Основные риски должны быть разрешены на ранних стадиях для того, чтобы повысить предсказуемость и избежать дорогостоящих дефектов и доработок, требующих возврата на более ранние стадии разработки.

3. **Изменяйте методы разработки с тем, чтобы перейти к *разработке, основанной на компонентах.*** Переход от склада ума, ориентированного на строки кода, к складу ума, ориентированному на компоненты, является необходимым для уменьшения количества кода, написанного человеком, и разработок на заказ. Компонент — это связный набор заранее написанных строк кода в исходном или выполняемом формате, для которого описаны интерфейс и поведение.

4. **Создавайте *среду для управления изменениями.*** Динамика итерационного процесса, включающего в себя параллельную работу различных команд над одними и теми же продуктами, влечет за собой необходимость объективного контроля над основными результатами.

5. **Повышайте легкость внесения изменений с помощью *инструментария, поддерживающего «круговую» разработку.*** «Круговая» разработка — это поддержка среды, необходимая для автоматизации и синхронизации рабочей информации в различных форматах (таких, как спецификации требований, проектные модели, исходный код, выполняемый код, тестовые варианты). Без существенной автоматизации учета использования системных ресурсов, управления изменениями, документирования и тестирования трудно вместить итерацию в приемлемые временные рамки, при которых внесение изменений приветствуется, а не избегается. Легкость внесения изменений — необходимая составляющая итерационного процесса, а создание интегрированной среды является критичным.

6. **Представляйте рабочие продукты проектирования в строгой, *основанной на моделях нотации.*** Подход, основанный на моделировании (например, UML), поддерживает развитие семантически богатых графических и текстуальных нотаций для описания проекта. Визуальное моделирование с использованием строгих нотаций и формального, пригодного для машинной обработки языка предоставляет гораздо больше средств достижения цели, чем традиционный подход просмотра и оценки проекта, представленного в виде бумажных документов, человеком.

7. **Обеспечивайте процесс инструментами для *объективного контроля за качеством* и оценки прогресса.** Необходимо включить в процесс оценку прогресса и качества всех промежуточных продуктов в течение всего жизненного цикла. Лучшими механизмами оценки являются хорошо описанные измерения, следующие непосредственно из рабочих продуктов и внедренные во все виды деятельности и команды.

8. **Используйте подход, *основанный на демонстрациях*, для оценки промежуточных продуктов.** Перевод рабочих продуктов, соответствующих текущему состоянию проекта (этими материалами могут быть ранний прототип, базовая архитектура или бета-версия некоторой функциональности), в выполняемую демонстрационную версию с соответствующими сценариями стимулирует раннее осуществление интеграции, более полное осознание компромиссов при разработке и раннее избавление от ошибок в архитектуре.

9. **Планируйте промежуточные версии для различных групп сценариев использования со *все возрастающим уровнем детализации.*** Существенным является то, что процесс управления созданием ПО приводит к ранним и постоянным демонстрациям в рамках функционального контекста системы (отдельных вариантов использования). Эволюция последовательных изменений проекта и его поколений должна быть соразмерна текущему уровню понимания. В этом случае связные сценарии использования оказываются главным механизмом для организации требований, описания содержания итерации, оценки реализаций и организации приемочного тестирования.

10. **Организуйте *конфигурируемый процесс*, который окажется экономически масштабируемым.** Никакой конкретный процесс не может быть пригоден для всех разработок ПО. Практический подход предполагает, что необходимо иметь возможность конфигурирования процесса для широкого спектра приложений. Процесс должен гарантировать экономию при больших масштабах и возврат инвестиций за счет использования общего духа самого процесса, широкой автоматизации и общих архитектурных образцов и компонентов.

Мои 10 наиглавнейших принципов не имеют научного обоснования. Однако они представляют собой сбалансированный взгляд на те повторяющиеся темы, которые представлены в настоящей книге. В таблице 4.1 приведено соответствие между тем, что я считаю 10-ю главными рисками традиционного процесса, и ключевыми атрибутами и принципами современного процесса. В таблице содержатся грубые обобщения, но она является введением в принципы современного процесса.

Таблица 4.1.
Подходы, используемые в современном процессе для решения проблем традиционного подхода

Традиционный процесс: 10 главных рисков	Влияет на	Современный процесс: внутренние особенности, позволяющие избавиться от рисков
1. Поздние поломки и избыточные дефекты/доработки	Качество, стоимость, сроки	Подход с упреждающей разработкой архитектуры Итерационная разработка Автоматизированное управление изменениями Процесс, противостоящий рискам
2. Увольнение ключевых сотрудников	Качество, стоимость, сроки	Успешное выполнение первых итераций Заслуживающие доверия управление и планирование
3. Неадекватные ресурсы для разработки	Стоимость, сроки	Среда как важнейшая составляющая процесса Предназначенная специально для производства интегрированная среда Продукты разработки, основанные на моделях «Круговая» разработка
4. Антагонизм между заинтересованными сторонами	Стоимость, сроки	Обзоры, основанные на демонстрациях Ориентированные на варианты использования требования/тестирование
5. Технологические проблемы	Стоимость, сроки	Подход с упреждающей разработкой архитектуры Компонентно-ориентированная разработка

Таблица 4.1. (продолжение)
Подходы, используемые в современном процессе для решения проблем традиционного подхода

Традиционный процесс: 10 главных рисков	Влияет на	Современный процесс: внутренние особенности, позволяющие избавиться от рисков
6. «Ползучее» изменение требований	Стоимость, сроки	Итерационная разработка Моделирование вариантов использования Обзоры, основанные на демонстрации
7. Паралич анализа	Сроки	Обзоры, основанные на демонстрации Ориентированные на варианты использования требования/тестирование
8. Ненадлежащая производительность	Качество	Оценка производительности, основанная на демонстрации Раннее установление обратной связи между архитектурой и производительностью
9. Чрезмерное внимание рабочим продуктам	Сроки	Оценка, основанная на демонстрации Объективный контроль качества
10. Ненадлежащее функционирование	Качество	Итерационная разработка Ранние прототипы, последовательные версии

4.3 ПЕРЕХОД К ИТЕРАЦИОННОМУ ПРОЦЕССУ

Современные процессы разработки ПО отошли от традиционной водопадной модели, в которой каждая следующая стадия зависит от завершения предыдущей. Возможны различные варианты, но обычно современные подходы требуют быстрого создания первоначальной версии системы на ранних этапах процесса разработки, в которой бы уделялось особое внимание высоким рискам, стабилизации базовой архитектуры и уточнению основных требований (с широким привлечением пользователей там, где это возможно). Дальнейшая разработка протекает как последовательность итераций, надстраивающих архитектурное ядро до тех пор, пока не будут достигнуты желаемые уровни функциональности, производительности и стабильности. (Эти итерации называют спиралью, последовательными шагами, поколениями или версиями.) Итерационный процесс имеет дело с системой в целом, а не с отдельными ее частями. Риск уменьшается на ранних стадиях жизненного цикла за счет непрерывной интеграции и уточнения требований, архитектуры и планов. Неожиданности, требующие возврата на более ранние стадии и являющиеся бичом традиционных проектов, исключаются.

Экономические преимущества, сопутствующие переходу от традиционной водопадной модели к итерационному процессу разработки, значительны, но трудно поддаются количественному определению. В качестве одной из возможных точек отсчета для вычисления ожидаемого экономического эффекта от улучшения процесса можно рассмотреть

экспоненциальные параметры процесса в модели COCOMO II (см. приложение B). Эта экспонента может находиться в диапазоне от 1.01 (практически отсутствует плата за большой масштаб) до 1.26 (существенная плата за большой масштаб). К параметрам, определяющим значение экспоненты процесса, относятся наличие прецедентов для приложений, гибкость процесса, разрешение архитектурных рисков, сплоченность команды и зрелость процесса создания ПО.

Ниже показано, какие параметры экспоненты процесса по COCOMO II соответствуют моим 10 наиважнейшим принципам современного процесса.

- **Наличие прецедентов приложений.** Опыт в конкретной области является критичным фактором для понимания того, как планировать и выполнять проект. Для систем, не имеющих прецедентов, одной из ключевых задач является определение рисков и создание прецедентов на ранних этапах, пусть даже они будут незаконченными или экспериментальными. Это одна из главных причин, по которой индустрия ПО перешла к ***итерационным процессам жизненного цикла.*** Итерации, выполняемые на ранних стадиях жизненного цикла, создают прецеденты, на основе которых продукт, процесс и планы могут разрабатываться со ***все возрастающей степенью детализации.***
- **Гибкость процесса.** Разработка современного ПО характеризуется таким широким пространством решений и таким большим количеством взаимосвязанных понятий, что возникает необходимость постоянного внесения изменений. Эти изменения могут быть присущи пониманию проблемы, пространству решений или планированию. Для ***эффективного управления изменениями*** рабочие продукты проекта должны иметь поддержку среды, соразмерную требованиям проекта. Как чересчур жесткие процессы, так и процессы с хаотическими изменениями обречены на провал, за исключением наиболее тривиальных случаев. Для достижения возврата инвестиций в ПО необходим ***настраиваемый процесс,*** который позволяет приспособить общую схему для целого спектра проектов.
- **Разрешение рисков, связанных с архитектурой.** В основе успешного итерационного процесса разработки лежит ***упреждающая разработка архитектуры.*** Команда разработчиков создает и добивается стабильности архитектуры до того, как приступить к разработке всего набора компонентов приложений. ***Подход с упреждающей разработкой архитектуры, основанный на компонентах,*** требует, чтобы инфраструктура, общие и управляющие механизмы были созданы на ранних стадиях жизненного цикла, и включает принятие решений о создании/покупке компонентов в процесс определения архитектуры. Такой подход инициирует начало интеграции на ранних стадиях жизненного цикла и верификации процесса разработки и продукта. Он также заставляет настроить и испытать среду разработки на ранних ее стадиях, способствуя таким образом раннему привлечению внимания к вопросам тестирования и оценок, ***основанных на демонстрации.***

- **Сплоченность команды.** Успешные команды сплочены, а сплоченные команды успешны. Нельзя с уверенностью утверждать, что является причиной, а что — следствием, но и сплоченные, и успешные команды имеют общие цели и приоритеты. Сплоченные команды избегают всего, что может стать источником проблем и роста энтропии проекта, являющегося результатом трудностей, которые возникают при попытке синхронизации требований различных заинтересованных сторон. Существует множество возможных причин для такого рода возмущений, но одной из основных является недостаток взаимодействия, в частности, обмен информацией исключительно посредством бумажных документов, представляющих информацию субъективно. Достижения технологии (такие, как языки программирования, UML и визуальное моделирование) привели к появлению более строгих и понятных нотаций для обмена информацией, касающейся разработки ПО, особенно в части материалов по требованиям и по разработке, которые раньше были узкоспециализированными и целиком основывались на обмене бумагами. Форматы, ***основанные на моделях***, оказались также способны поддерживать ***«круговую» разработку***, необходимую для облегчения внесения изменений, что требуется для развития представлений проекта.
- **Зрелость процесса создания ПО.** В качестве общепринятой точки отсчета при оценке процесса создания ПО используется предложенная Институтом программной инженерии (SEI) модель технологической зрелости (Capability Maturity Model, CMM). Подобно тому, как опыт в конкретной области знаний критичен и позволяет избегать рисков и использовать ценные качества и полученные уроки, так и технологическая зрелость процесса создания ПО критична и позволяет избегать рисков при разработке ПО и использовать ценные качества и полученные уроки в рамках данной организации. («За» и «против» CMM подробно обсуждаются в приложении Е.) Одной из ключевых тем данной книги является то, что действительно зрелый процесс может возникать только в рамках интегрированной среды, которая обеспечивает адекватный уровень автоматизации процесса для ***объективного контроля качества***.

Часть II

Основы управления процессом создания ПО

Глава 5 ◆ **Стадии жизненного цикла**

Глава 6 ◆ **Рабочие продукты процесса**

Глава 7 ◆ **Архитектура ПО, основанная на моделях**

Глава 8 ◆ **Рабочие процессы**

Глава 9 ◆ **Контрольные точки процесса**

Стандартизация в рамках обычного процесса — смелое начинание для любой организации, занимающейся ПО. Она включает в себя широкий спектр необходимых действий. Процесс, рекомендуемый в этой книге, содержит лишь несколько специальных стандартов: стадии жизненного цикла, рабочие продукты жизненного цикла, рабочие процессы в рамках жизненного цикла и контрольные точки жизненного цикла. Эти элементы являются определяющими при переходе от традиционного подхода к итерационному. Я наблюдал ситуации, когда организации пытались сделать меньше (стандартизация недостаточная или вообще отсутствует) или больше (чрезмерная стандартизация), что не позволяло повысить возврат инвестиций в ПО. Стандартизация процесса требует взвешенного подхода.

В главах, входящих в часть II, описываются основы современного, итерационного процесса управления созданием ПО: стадии жизненного цикла, рабочие продукты, рабочие процессы и контрольные точки. Упреждающая разработка архитектуры — ключевая тема, которая объединяет все эти главы. С точки зрения управления как сама архитектура, так и то особое внимание, которое должно уделяться ей, являются жизненно важными для успеха подхода в целом.

Глава 5

Стадии жизненного цикла

Ключевые моменты

- Стадия разработки в рамках жизненного цикла включает в себя планы, требования и архитектуру, что позволяет разрешать связанные с разработкой риски. Эта стадия завершается созданием реализуемой базовой архитектуры.
- На стадии производства в рамках жизненного цикла создаются работающие версии ПО, реализующие некоторую функциональную возможность в контексте основных планов, требований и архитектуры, полученных на стадии разработки.

Характеристикой, в наибольшей степени определяющей успешность процесса создания ПО, является четко определенное разграничение между работами, направленными на «исследования и разработку», и работами, направленными на «производство». Главной причиной того, что проекты не завершаются успехом, обычно оказывается неудача при попытке четко определить и выполнить эти две стадии, сохранить между ними правильное соотношение и распределить усилия подходящим образом. Это утверждение является справедливым как для традиционного, так и для итерационного процесса. Многие проекты, завершившиеся неудачей, обладают одной из следующих характеристик:

- Чрезмерное внимание уделяется исследованиям и разработке. Выполняется слишком много анализов или бумажных исследований, либо все время откладывается формирование основных проектных решений. Такой перекос типичен для традиционного процесса создания ПО и провоцируется им.
- Чрезмерное внимание уделяется изготовлению. Типичными в этом случае являются принятие спонтанных решений, преждевременная работа чересчур усердных программистов и постоянное хакерство.

Современные успешные проекты — и даже те, что разработаны с помощью традиционного процесса, — обычно имеют хорошо обозначенную основную контрольную точку в осуществлении проекта, по достижении которой происходит осязаемый переход от состояния исследований к состоянию производства. Более ранние стадии сосредоточены на обретении функциональных возможностей. Более поздние стадии касаются получения продукта, который может быть передан заказчику. При этом особое внимание уделяется стабильности, производительности, подгонке и срокам завершения. Такое равновесие в рамках жизненного цикла трудноуловимо и по-прежнему практически неосязаемо, но является одним из оснований успешного управления проектом по созданию ПО.

По определению, современный процесс разработки ПО должен поддерживать:

- Эволюцию планов, требований и архитектуры вместе с четко определенными точками синхронизации
- Управление рисками и объективные способы измерения прогресса и качества
- Эволюцию системы с помощью демонстрации все возрастающих функциональных возможностей

5.1 СТАДИИ РАЗРАБОТКИ И ИЗГОТОВЛЕНИЯ

Экономические соображения, приведенные в предыдущих главах, дают основу для описания жизненного цикла. Для получения экономии при больших масштабах и высокой отдачи от инвестиций нам необходимо двигаться в направлении процесса производства ПО, ведомого технологическими усовершенствованиями в области автоматизации процесса и разработки, основанной на компонентах. В первом приближении существуют две стадии жизненного цикла:

1. Стадия разработки, реализуемая менее предсказуемыми, но меньшими по размеру группами, которые выполняют работы по проектированию и синтезу.
2. Стадия изготовления, реализуемая более предсказуемыми, но большими по размеру группами, которые выполняют работы по созданию ПО, тестированию и вводу в действие.

В таблице 5.1 показано, чему уделяется внимание на этих двух стадиях.

Переход от разработки к изготовлению является критичным событием для различных исполнителей. Планы изготовления согласованы, и существует достаточно полное понимание проблемы и путей ее решения, которыми все исполнители могут твердо руководствоваться при производстве. В зависимости от специфики проекта, в частности от ключевых определителей (см. ниже), время и ресурсы, выделяемые на эти две стадии, могут сильно варьироваться.

Таблица 5.1.
Две стадии жизненного цикла: разработка и изготовление

Аспект жизненного цикла	Чему уделяется внимание на стадии разработки	Чему уделяется внимание на стадии изготовления
Уменьшение риска	Сроки, техническая осуществимость	Стоимость
Продукты	Базовая архитектура	Основные версии продукта
Виды работ	Анализ, проектирование, планирование	Реализация, тестирование
Оценка	Демонстрация, проверка, анализ	Тестирование
Экономика	Борьба с платой за большой масштаб	Экономия при больших масштабах
Управление	Планирование	Операции

Разбиение жизненного цикла всего на две стадии выглядит излишне грубым и примитивным для большинства приложений. Соответственно стадия разработки делится на две отдельные стадии – начальную стадию и стадию уточнения (детального проектирования), а стадия изготовления – на стадии конструирования (реализации) и ввода в действие. Эти четыре стадии жизненного цикла проекта могут быть поставлены в соответствие концептуальной схеме спиральной модели [Boehm, 1988], как показано на рис. 5.1. Их названия обозначают состояние проекта. Размер спирали соответствует инерции проекта с учетом масштаба и детальности рабочих продуктов, которые созданы к этому моменту. Инерция проявляется в обеспечении непротиворечивости рабочих продуктов, регрессионном тестировании, документировании, анализе качества и контроле конфигурации. Возросшая инерция может иметь очень небольшое или, по крайней мере, очень простое влияние на изменение каждого отдельного компонента или вида деятельности. Однако время реакции, необходимое для приспособления к существенным изменениям в архитектуре, существенным изменениям в требованиях, существенным сдвигам в планах либо существенным организационным пертурбациям, возрастает для последующих стадий.

Для большинства традиционных жизненных циклов стадии получают название по главному виду деятельности, осуществляемому в рамках каждой стадии: анализ требований, разработка, кодирование, тестирование модулей, интеграционное тестирование, тестирование системы. При разработке ПО традиционным способом особое внимание уделяется в основном последовательности процесса, когда требуется сначала завершить один вид деятельности, прежде чем начать следующий.

В случае итерационного процесса каждая стадия включает в себя все виды деятельности в разных пропорциях. Уровень различных видов деятельности на этих стадиях обсуждается в главе 8. Главные цели, основные виды деятельности и общие критерии оценок для каждой стадии рассматриваются ниже.

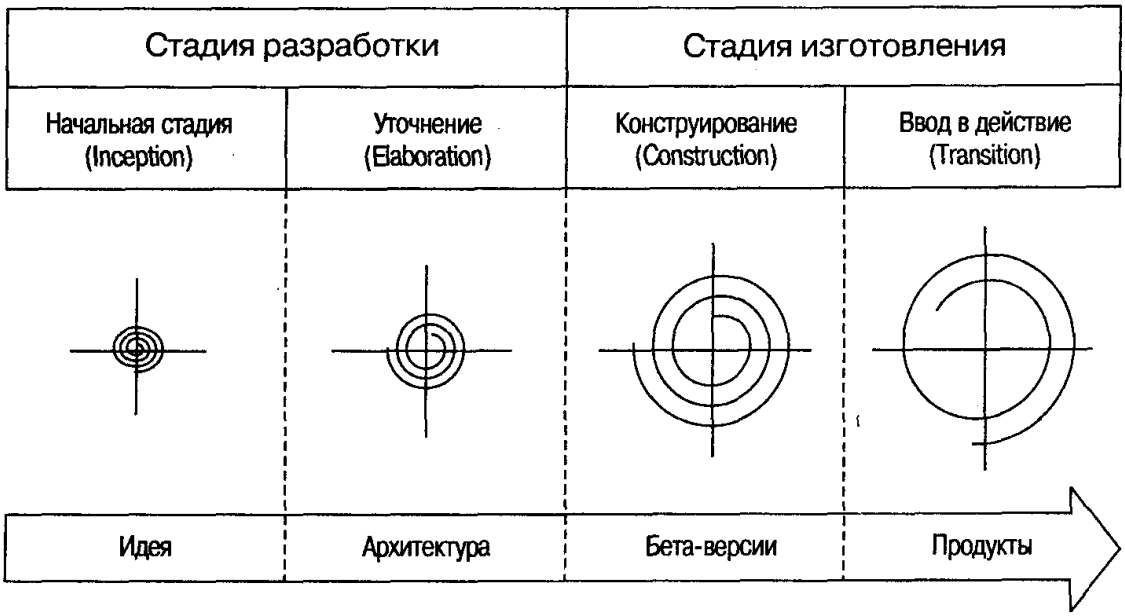


Рис. 5.1. *Стадии жизненного цикла процесса*

5.2 НАЧАЛЬНАЯ СТАДИЯ

Первоочередной задачей начальной стадии является достижение согласованности между заинтересованными сторонами относительно целей всего жизненного цикла проекта.

Главные цели

- Определение области действия проекта и граничных условий, включая рабочую концепцию, критерии приемки и ясное понимание того, чему следует и чему не следует быть в продукте
- Выявление критичных вариантов использования системы и базовых сценариев, которые будут определяющими для основных проектных решений
- Демонстрация по крайней мере одной возможной архитектуры для некоторых из основных сценариев
- Примерная оценка стоимости и сроков выполнения всего проекта (включая подробные оценки для стадии уточнения)
- Определение потенциальных рисков (источников непредсказуемости)

Основные виды деятельности

- Определение области действия проекта. Этот вид деятельности включает в себя фиксацию требований и рабочей концепции, описывающей взгляд пользователя на требования, в проектном репозитории. Репозиторий должен обладать достаточностью для определения проблемной области и выработки критериев приемки для конечного продукта.

- Синтез архитектуры. Оцениваются проектные решения, неоднозначности в проблемной области, а также наличие доступных для данной области технологий и готовых компонентов. Создается такой репозиторий, который пригоден для демонстрации реализуемости по крайней мере одной возможной архитектуры и начальной основы для принятия решений по вопросам о покупке/изготовлении с тем, чтобы можно было получить оценки стоимости, сроков и ресурсов.
- Планирование и подготовка бизнес-плана. Оцениваются альтернативы для управления рисками, найма персонала, планов по итерациям и компромиссы по стоимости/срокам/прибыльности. Определяется инфраструктура (инструменты, процессы, поддержка), требующаяся для поддержки задач разработки в течение всего жизненного цикла.

Основные критерии оценки

- Все ли заинтересованные лица согласны с определением области действия и с оценками стоимости и сроков?
- Понятны ли требования (это может быть определено по правильности критичных вариантов использования)?
- Являются ли оценки стоимости и сроков, приоритеты, риски и процессы разработки заслуживающими доверия?
- Соответствуют ли масштаб и детальность архитектурного прототипа предварительным критериям? (Главной целью создания прототипа предполагаемой архитектуры является получение механизма для понимания области действия и оценки того, насколько можно доверять группе разработки при решении частных технических проблем.)
- Действительно ли реальные затраты ресурсов приемлемы по сравнению с планируемыми затратами?

5.3 СТАДИЯ УТОЧНЕНИЯ

Легко доказать, что стадия уточнения является наиболее критичной из всех четырех стадий. По ее окончании «разработка» считается завершенной, и должно быть принято решение, переходить или не переходить к стадиям производства. Для большинства проектов это решение соответствует переходу от «легких» действий с малым финансовым риском к действиям с более высоким финансовым риском и существенной инерцией. В то время как процесс должен постоянно учитывать вносимые в него изменения, в результате выполнения стадии уточнения необходимо получить гарантию того, что архитектура, требования и планы достаточно стабильны, что вероятность рисков снижена до подходящего уровня, а стоимость и сроки завершения разработки могут быть предсказаны с приемлемой точностью. Концептуально этот уровень правильности

должен соответствовать тому, что необходимо для данной организации, это позволит перейти к стадии конструирования, затраты на которую могут быть определены заранее.

В одну или больше итераций, из которых состоит стадия уточнения, включается создание выполняемого прототипа, зависящего от области действия, размера, риска и новизны проекта. Эти усилия направляются, по крайней мере, на критичные варианты использования, которые определены на начальной стадии и обычно предполагают наличие самых высоких рисков в проекте. Хотя прототипирование компонентов, определяющих качество продукта, является постоянной целью, это не исключает разработку одного или нескольких экспериментальных прототипов для снижения конкретных рисков, таких как компромиссы между проектированием и требованиями, анализ возможности создания компонентов или демонстрация инвесторам.

Главные цели

- Определение базовой архитектуры настолько быстро, насколько это практически возможно (создание «моментального снимка» текущей конфигурации, в которой все изменения учтены, отслежены и поддерживаются)
- Определение базовой концепции продукта
- Определение (с высокой точностью) плана стадии конструирования
- Демонстрация того, что базовую архитектуру можно будет привести в соответствие с общей концепцией за разумную стоимость и за разумное время

Основные виды деятельности

- Уточнение концепции. Этот вид деятельности связан с достижением правильного понимания критичных вариантов использования, от которых зависят решения по архитектуре и планированию.
- Уточнение процесса и инфраструктуры. Определяются процесс конструирования, инструментарий и средства автоматизации процесса, а также промежуточные контрольные точки и соответствующие им критерии оценки.
- Уточнение архитектуры и выбор компонентов. Оцениваются потенциальные компоненты, и выясняется, какие решения следует принимать по вопросу их приобретения/создания с тем, чтобы достоверно определить стоимость и сроки стадии конструирования. Выбранные компоненты архитектуры объединяются и оцениваются относительно основных сценариев. Уроки, извлеченные в результате этой деятельности, вполне могут привести к изменению всей архитектуры, выполняемому в результате рассмотрения альтернативных вариантов, или к переопределению требований.

Основные критерии оценки

- Является ли общая концепция стабильной?
- Является ли стабильной архитектура?
- Видно ли из демонстрационной версии, что на основные источники рисков обращено внимание и что эти риски надежно разрешены?
- Достаточно ли качественен план стадии конструирования, и подкреплен ли он заслуживающими доверия оценками?
- Все ли заинтересованные лица согласны с тем, что имеющаяся концепция может быть реализована при выполнении существующего плана в виде полностью завершенной системы в контексте заданной архитектуры?
- Являются ли реальные затраты ресурсов приемлемыми по сравнению с планируемыми затратами?

5.4 СТАДИЯ КОНСТРУИРОВАНИЯ

На стадии конструирования все компоненты и функциональные возможности интегрируются в одно приложение, и все функциональные возможности тщательно тестируются. По мере необходимости интегрируется также вновь разработанное ПО. Стадия конструирования представляет собой процесс производства, в котором особое внимание уделяется управлению ресурсами и контролю за выполнением с целью оптимизации стоимости, сроков и качества. В этом смысле общая направленность менеджмента претерпевает изменения, эволюционируя от разработки интеллектуальной собственности в течение начальной стадии и стадии уточнения в сторону разработки коммерческих продуктов в процессе конструирования и ввода в действие.

Многие проекты оказываются настолько большими, что в них параллельно выполняется множество работ. Параллельные работы способны значительно ускорить получение готовых версий, однако они могут также усложнить управление ресурсами и синхронизацию рабочих процессов и групп. Стабильная архитектура тесно связана с хорошо понимаемым планом. Другими словами, одним из критичных качеств любой архитектуры является простота ее создания. Это та причина, по которой на стадии уточнения уделяется особое внимание сбалансированности разработки архитектуры и плана.

Главные цели

- Минимизация стоимости разработки за счет оптимизации использования ресурсов и исключения излишних дефектов и доработок
- Достижение требуемого качества настолько быстро, насколько это практически осуществимо

- Создание полезных версий (альфа-, бета- и других тестовых версий) настолько быстро, насколько это практически осуществимо

Основные виды деятельности

- Управление использованием ресурсов, контроль и оптимизация процесса
- Завершение разработки компонентов и тестирования на соответствие критериям оценки
- Оценка версий продукта по критериям приемки, соответствующим концепции продукта

Основные критерии оценки

- Является ли основа продукта достаточно сформировавшейся для передачи его пользователям? (Наличие дефектов не является препятствием для достижения цели следующей версии.)
- Является ли основа продукта достаточно стабильной для передачи его пользователям? (Незавершенные изменения не являются препятствием для достижения цели следующей версии.)
- Готовы ли заинтересованные стороны к передаче продукта пользователям?
- Являются ли реальные затраты ресурсов приемлемыми по сравнению с планируемыми затратами?

5.5 СТАДИЯ ВВОДА В ДЕЙСТВИЕ

Стадия ввода в действие наступает, когда основа продукта оказывается достаточно сформировавшейся для установки системы у конечных пользователей. Для этого обычно требуется, чтобы применяемое подмножество системы было выполнено с приемлемым уровнем качества и пользовательской документацией, тогда передача приведет к положительным результатам. Эта стадия может включать в себя любые из следующих видов деятельности:

1. Бета-тестирование с целью приведения системы в соответствие с ожиданиями пользователя
2. Бета-тестирование и параллельная эксплуатация вместе с существующей заменяемой (legacy) системой
3. Конвертирование эксплуатируемых баз данных
4. Обучение пользователей и службы сопровождения

Стадия ввода в действие заканчивается, когда продукт оказывается полностью внедренным. Для некоторых проектов окончание жизненного цикла может совпадать с отправной точкой следующей версии

проекта. В других случаях оно может совпадать с окончательной передачей всей информации третьей стороне, ответственной за эксплуатацию, *сопровождение и развитие.*

Основное внимание на стадии ввода в действие уделяется видам деятельности, необходимым для передачи ПО в руки пользователей. Обычно эта стадия состоит из нескольких итераций, включая бета-версии, общедоступные версии, а также версии с исправленными ошибками и расширениями. Значительные усилия затрачиваются на создание документации, ориентированной на пользователя, на подготовку пользователей, поддержку пользователей в начальный период применения продукта и на установление обратной связи с пользователями. (На этом этапе жизненного цикла обратную связь с пользователями следует ограничить в основном настройкой продукта, изменением конфигурации, установкой и вопросами эксплуатации.)

Главные цели

- Достижение самостоятельной поддержки системы со стороны пользователей
- Достижение согласия между заинтересованными сторонами относительно того, что внедряемый продукт завершен и не противоречит *критериям оценки, определенным в концепции*
- Создание окончательной версии продукта настолько быстро и настолько дешево, насколько это осуществимо на практике

Основные виды деятельности

- Синхронизация и интеграция параллельно созданных частей системы в единое целое
- Специальные работы, связанные с внедрением системы (перевод в новую среду эксплуатации, коммерческое оформление и производство, разработка стандартного комплекта поставки, обучение специалистов)
- Оценка основных параметров внедрения в соответствии с общей концепцией и критериями приемки из набора требований

Основные критерии оценки

- Удовлетворены ли пользователи?
- Являются ли реальные затраты ресурсов приемлемыми по сравнению с планируемыми затратами?

Каждая их четырех стадий состоит из одной или большего числа итераций, в которых какая-либо из технических возможностей создается в виде, пригодном для демонстрации, и оценивается по некоторому набору критериев. Итерация (см. главу 8) представляет собой последовательность действий, для которых существует четко определенное

промежуточное событие (контрольная точка; см. главу 9); рамки и результаты каждой итерации фиксируются посредством отдельных рабочих продуктов (см. главу 6). В то время как для основных контрольных точек по окончании каждой итерации применяются формальные (принятые заинтересованными лицами) варианты критериев оценки и описаний версий, для второстепенных контрольных точек используются неформальные (контролируемые на уровне организации) варианты рабочих продуктов. Каждой стадии соответствует завершение некоторого числа итераций, достаточного для достижения заданного общего состояния проекта. Переход от одной стадии к последующей больше напоминает принятие определяющего решения, чем завершение того или иного этапа разработки ПО. Эти внутренние переходы от одной стадии к другой являются главными анкерными точками процесса создания ПО, когда приводятся в соответствие технические и управленческие перспективы, а между всеми исполнителями достигается согласие на базе текущего понимания требований, разработки и планов по завершению.

Глава 6

Рабочие продукты процесса

Ключевые моменты

- Рабочие продукты процесса подразделяются на пять видов: управление, требования, проектирование, реализация и внедрение.
- Рабочие продукты управления содержат информацию, необходимую для синхронизации потребностей заинтересованных сторон.
- Рабочие продукты требований, проектирования и внедрения ведутся в строгой нотации, которая допускает автоматизированный анализ и просмотр.

В традиционных проектах по созданию ПО основное внимание уделяется последовательной разработке рабочих продуктов: сформулируйте требования, сконструируйте удовлетворяющую этим требованиям проектную модель, создайте соответствующую этой модели реализацию, а затем скомпилируйте и протестируйте реализацию для ее внедрения. Такой процесс подходит для небольших, выполняемых исключительно на заказ разработок, в которых представление на этапе проектирования, представление на этапе реализации и представление на этапе внедрения тесно связаны. Например, определенная программа, предназначенная для выполнения на определенном компьютере определенного типа и состоящая целиком из компонентов специального назначения, созданных на заказ, может строиться таким образом, чтобы все представления вытекали одно из другого.

Однако такой подход не годится для современных программных систем, в которых системная сложность (по многим параметрам) приводит к такому бесчисленному количеству рисков и слабой взаимосвязи, что эффективно использовать упрощенный подход последовательных преобразований не представляется возможным. Большинство современных систем состоит из огромного числа компонентов (некоторые из них сделаны на заказ, другие предназначены для повторного использования, третьи являются коммерческим продуктом), способных работать в разнородной

сети распределенных платформ. Для них требуются иные последовательности эволюции рабочих продуктов и совершенно иной подход к преемственности.

За последние 20 лет индустрия ПО достигла определенного уровня зрелости, а процесс управления стал итерационным. Вместо того чтобы создавать рабочие продукты последовательно, со всеми ними работают одновременно, а ограничения, различия в уровнях абстракции и степени свободы находятся в равновесии. Повторяющиеся в успешных проектах особенности показывают, что рабочие продукты развиваются вместе с уровнями детализации. Они не проходят однонаправленное линейное преобразование от требований к проектированию, реализации и внедрению. Выбор, касающийся реализации и внедрения, оказывает влияние на то, в каком виде формулируются требования, и на то, в каком направлении происходит проектирование. Информация и решения могут перетекать различными способами из одних рабочих продуктов в другие. Задача хорошего процесса разработки — исключить ненужные, непродуманные ограничения и приспособиться к реально существующим в разработке ограничениям.

Каково же влияние итерационного подхода на рабочие продукты? Главным отличием от традиционного подхода является то, что внутри каждой стадии жизненного цикла работа ни в одном направлении не развивается лишь поступательно, так же как создание рабочих продуктов не происходит монотонно от одних продуктов к другим. Напротив, различные виды деятельности направлены на повторное обращение к одним и тем же рабочим продуктам, что позволяет последовательно обогащать общее описание системы и сам процесс уроками, извлеченными при попытке сохранения равновесия между широтой и глубиной информации.

6.1 КОМПЛЕКТЫ РАБОЧИХ ПРОДУКТОВ

Для того чтобы сделать процесс разработки программной системы управляемым, различные виды информации разбиваются на комплекты рабочих продуктов. Каждый комплект состоит из соответствующих продуктов, набор которых остается неизменным и которые имеют единый формат представления (такой, как текст на английском языке, код C++, Visual Basic, Java, стандартный шаблон документа, стандартный шаблон электронной таблицы или UML-модель). В то время как *комплект* — это законченный аспект системы, отдельные *продукты* представляют собой связную информацию, которая разрабатывается и рассматривается как единая сущность. Для каждой конкретной организации, проекта или системы некоторые из этих продуктов — и даже некоторые из комплектов — могут оказаться несущественными или ненужными. В целом, однако, в каждом комплекте должна содержаться некоторая информация с тем, чтобы все заинтересованные стороны были удовлетворены.

Продукты жизненного цикла ПО организованы в виде пяти отдельных комплектов, грубо их можно разделить по тому языку, на котором написаны документы, входящие в комплект: управление (специальные текстовые форматы), требования (организованный текст и модели

проблемной области), проектирование (модели области решений), реализация (понятный человеку язык программирования и соответствующие исходные файлы) и внедрение (машинные языки и соответствующие файлы).

Появление строгих и более мощных нотаций для описания требований и проекта, которые поддерживают упреждающую разработку архитектуры, явилось большим технологическим достижением. В частности, Unified Modeling Language превратился в удобный формат представления, а именно, в визуальные модели с четко определенными синтаксисом и семантикой для рабочих продуктов, связанных с требованиями и проектированием. Визуальное моделирование с использованием UML — это простая нотация для рабочих продуктов ранних этапов жизненного цикла. Комплекты продуктов показаны на рис. 6.1; их цели и нотации обсуждаются ниже.

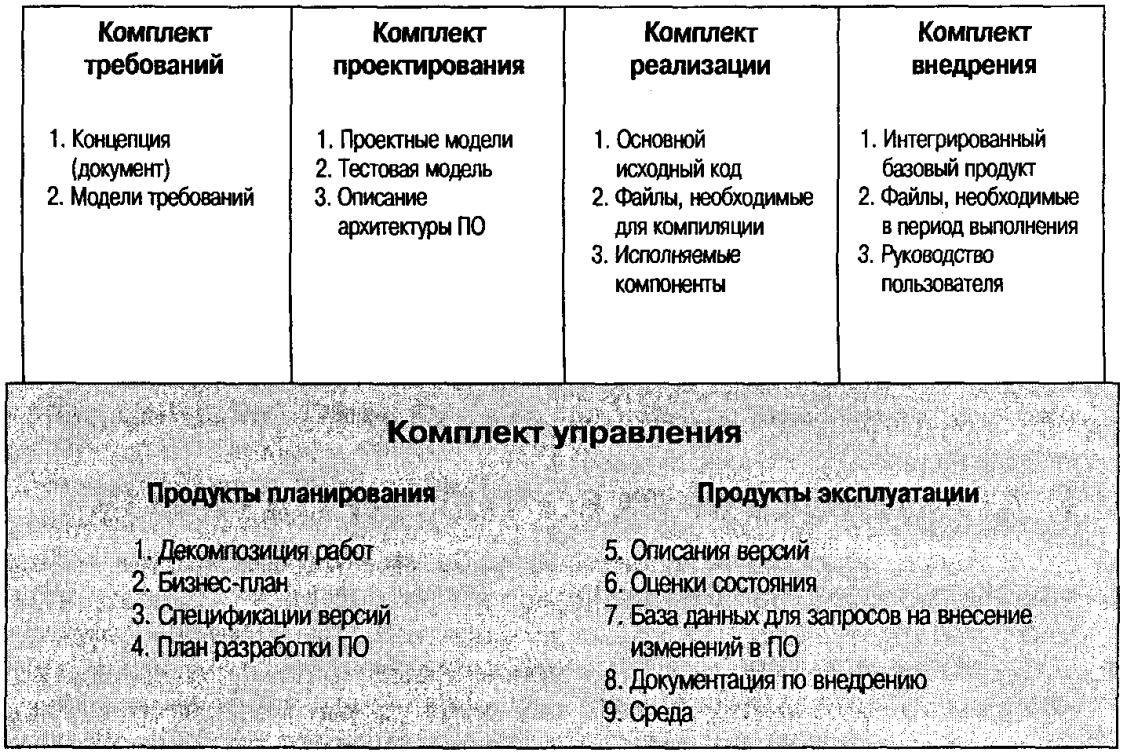


Рис. 6.1. *Обзор комплектов продуктов*

6.1.1 Комплект управления

Комплект управления содержит рабочие продукты, связанные с планированием процесса и его осуществлением. В этих продуктах используются специализированные нотации, включая текст, графику или любое другое представление, необходимое для того, чтобы зафиксировать принятые соглашения среди сотрудников, работающих над проектом (менеджеры проекта, архитекторы, разработчики, тестировщики, специалисты по маркетингу, администраторы), среди заинтересованных лиц

(ответственный за финансирование, пользователь, менеджер проекта по созданию ПО, менеджер организации, регулирующий орган), а также между сотрудниками, работающими над проектом, с одной стороны, и заинтересованными лицами, с другой. Конкретные продукты, включаемые в этот комплект, состоят из декомпозиции работ (детальный план работ и механизм контроля за финансированием), бизнес-плана (стоимость, сроки, ожидаемая прибыль), спецификации версии (область действия, план, основные задачи версии), плана разработки ПО (вариант процесса, используемого в проекте), описания версии (описание основных результатов версии), оценки состояния (периодическое определение хода выполнения проекта), запросов на внесение изменений (описание отдельных изменений), документации внедрения (план перевода в новую среду эксплуатации, курс обучения, набор для поставки) и среды (программный и аппаратный инструментарий, автоматизация процесса, документация, дополнительные курсы обучения, которые необходимы для осуществления процесса, описанного в плане разработки ПО, и для создания рабочих продуктов).

Рабочие продукты из комплекта управления оцениваются и измеряются комбинацией следующих пунктов:

- Рассмотрение соответствующими заинтересованными сторонами
- Анализ изменений, т.е. отличий текущей версии продукта от предыдущей версии (тенденции управления и тенденции изменения хода выполнения проекта в терминах стоимости, сроков и качества)
- Демонстрации сбалансированности всех продуктов по достижении основных контрольных точек, в частности, тщательности проработки бизнес-плана и общей концепции

6.1.2 Комплекты разработки

Комплекты разработки состоят из комплекта требований, проектного комплекта, комплекта реализации и комплекта внедрения. Основным механизмом для оценки качества каждого комплекта является преобразование информации из одного комплекта в другой, что позволяет добиться сбалансированного восприятия рабочих продуктов требований, проектирования, реализации и внедрения. Каждый из этих компонентов описания системы изменяется с течением времени.

Комплект требований

Для описания общей концепции системы используется структурированный текст, позволяющий документировать область действия проекта, в основе которого лежит контракт между ответственным за финансирование и группой разработчиков. Для дополнительных спецификаций могут также использоваться специализированные форматы (например, законодательные требования), а также пользовательские макеты или другие прототипы, в которых содержатся требования. Для рабочего представления моделей требований используется нотация UML (модели вариантов

использования, модели предметной области). Комплект требований является главным рабочим контекстом для определения трех других комплектов, касающихся разработки, и именно на его основе формируются тестовые варианты.

Рабочие продукты из комплекта требований оцениваются и измеряются комбинацией следующих пунктов:

- Анализ непротиворечивости по отношению к спецификациям версий из комплекта управления
- Анализ непротиворечивости между общей концепцией и моделями требований
- Сравнение с комплектами проектирования, реализации и внедрения для определения непротиворечивости, полноты и смыслового равновесия между информацией из различных комплектов
- Анализ изменений в текущей версии рабочих продуктов требований по сравнению с предыдущими версиями (тенденции по уменьшению количества дефектов и доработок)
- Субъективное рассмотрение других составляющих качества

Комплект проектирования

В проектных моделях, описывающих получаемые решения, используется язык UML. В комплекте проектирования представлены различные уровни абстракции, которые соответствуют различным компонентам из области решений (их индивидуальные свойства, атрибуты, статические связи и динамические взаимодействия). В этих моделях содержится достаточное количество информации о структуре и поведении для определения спецификации рабочих продуктов (количество и спецификации составных частей и материалов, затраты труда и другие прямые затраты). Информация, содержащаяся в проектных моделях, может быть напрямую, зачастую автоматически, переведена в подмножество продуктов, относящихся к комплектам реализации и внедрения. Отдельные продукты из комплекта проектирования включают в себя проектную модель, тестовую модель и описание архитектуры ПО (ту часть информации из проектной модели, которая имеет отношение к описанию архитектуры).

Рабочие продукты проектирования оцениваются и измеряются комбинацией следующих пунктов:

- Анализ внутренней непротиворечивости и качества проектной модели
- Анализ непротиворечивости по отношению к моделям требований
- Перевод в комплекты реализации и внедрения и соответствующие нотации (например, трассировка, генерация исходного кода, компиляция, редактирование связей) для определения непротиворечивости, полноты и смыслового соответствия между информацией из различных комплектов

- Анализ изменений в текущей версии проектной модели по сравнению с предыдущими версиями (тенденции по уменьшению количества дефектов и доработок)
- Субъективное рассмотрение других составляющих качества

Поскольку степень автоматизации анализа проектных моделей в настоящее время является ограниченной, следует полагаться на анализ, выполняемый человеком. Такая ситуация должна измениться в течение нескольких ближайших лет с формированием инструментария для анализа проектных моделей, который поддерживает сбор значений множества параметров, анализ сложности, анализ стиля, эвристический анализ и анализ непротиворечивости.

Комплект реализации

Комплект реализации включает в себя исходный код (запись на языке программирования), который представляет собой реализации компонентов (их форму, интерфейс и зависимости), и все требуемые для автономного тестирования компонентов исполняемые файлы. Исполняемые файлы являются простыми составными частями, необходимыми для создания конечного продукта, включая компоненты, созданные на заказ, программные интерфейсы (API) коммерческих компонентов, а также API компонентов повторного использования или компонентов, имеющихся в исходном языке программирования (например, в Ada 95, C++, Visual Basic, Java или Assembly). Рабочие продукты комплекта также можно транслировать (скомпилировать и отредактировать связи) в подмножество комплекта внедрения (исполняемые файлы в окончательном виде). Конкретные рабочие продукты включают в себя самодокументируемый исходный код продукта и связанные с ним файлы (сценарии компиляции, инфраструктура для управления конфигурацией, файлы с данными), самодокументируемый тестовый исходный код и связанные с ним файлы (файлы с входными данными для тестирования, файлы с результатами тестирования), исполняемые файлы для независимого запуска компонентов и файлы для проведения тестирования компонентов.

Комплекты реализации имеют форматы, пригодные для чтения человеком; оцениваются и измеряются комбинацией следующих пунктов:

- Анализ непротиворечивости по отношению к проектным моделям
- Перевод в нотации комплекта внедрения (например, компиляция и редактирование связей) для определения непротиворечивости и полноты различных комплектов рабочих продуктов
- Оценка исходных или исполняемых файлов компонентов на предмет соответствия подходящим критериям оценки с помощью проверок, анализа, демонстраций или тестирования
- Выполнение тестовых вариантов для независимых компонентов с автоматическим сравнением ожидаемых результатов с полученными.

- Анализ изменений в текущей версии комплекта реализации по сравнению с предыдущими версиями (тенденции по уменьшению количества дефектов и доработок)
- Субъективное рассмотрение других составляющих качества

Комплект внедрения

Комплект внедрения содержит файлы, поставляемые пользователю, записи на машинном языке, исполняемое ПО, сценарии сборки, сценарии инсталляции и данные, необходимые для использования продукта в той среде, для которой он предназначен. Записи на машинных языках представляют компоненты продукта в конечном виде, служащем для распространения среди пользователей. Содержимое комплекта внедрения может быть инсталлировано, выполнено в соответствии со сценариями использования (протестировано) и динамически перенастроено для поддержки тех свойств, которые должны присутствовать в конечном продукте. Конкретные рабочие продукты состоят из основных исполняемых файлов и связанных с ними файлов, которые могут потребоваться в период выполнения, а также из руководств пользователя.

Комплекты внедрения оцениваются и измеряются комбинацией следующих пунктов:

- Тестирование сценариев использования и характеристик качества, определенных в комплекте требований, для оценки непротиворечивости, полноты и смыслового соответствия между информацией, содержащейся в этих двух комплектах
- Тестирование стратегий распределения, репликации и размещения на предмет соответствия компонентов комплекта реализации физическим ресурсам среды внедрения (тип платформы, количество платформ, топология сети)
- Тестирование на предмет соответствия описанных в руководстве пользователя сценариев использования, таких как инсталляция, динамическое изменение конфигурации, основное применение и управление в аномальных ситуациях
- Анализ изменений в текущей версии комплекта внедрения по сравнению с предыдущими версиями (тенденции уменьшения количества дефектов, изменения в производительности)
- Субъективное рассмотрение других составляющих качества

Выбор именно комплектов управления, требований, проектирования, реализации и внедрения не является научно обоснованным. Основная задача при определении комплектов — оптимизация представления процессов, рабочих продуктов и целей процесса. Некоторые обоснования, приведшие к данному концептуальному подходу, описываются ниже. Могут существовать незначительные исключения из этих представлений, тем не менее они оказываются весьма полезными для общего понимания комплектов рабочих продуктов.

В каждом комплекте рабочих продуктов используются свои нотации для представления соответствующих рабочих продуктов. Нотации комплекта управления (специализированный текст, графика, варианты использования) позволяют выразить планы, процесс, цели и критерии приемки. Нотации комплекта требований (структурированный текст и UML-модели) определяют контекст разработки и эксплуатации. Нотации комплекта проектирования (в UML) содержат схемы и диаграммы (проектирование архитектуры и компонентов). Нотации комплекта реализации (языки программирования) обеспечивают создание строительных блоков системы в виде, пригодном для чтения человеком. Нотации комплекта внедрения (исполняемые файлы и файлы с данными) содержат полное решение в машинных форматах.

Каждый из комплектов рабочих продуктов является преобладающим при разработке на протяжении одной стадии жизненного цикла; остальные комплекты играют контролирующую и балансирующую роли. Как показано на рис. 6.2, каждый комплект рабочих продуктов является доминирующим на протяжении какой-либо одной из стадий жизненного цикла: комплект требований сосредоточен на начальной стадии, проектирования — на стадии уточнения, реализации — на стадии конструирования, внедрения — на стадии ввода в действие. Рабочие продукты управления также подвергаются изменениям на протяжении жизненного цикла, но уровень этих изменений остается практически постоянным.

Большинство из современных инструментов разработки ПО почти точно соответствует одному из пяти комплектов рабочих продуктов.

1. Управление: инструменты планирования, управления рабочими процессами, отслеживания дефектов, управления изменениями, документирования, работы с электронными таблицами, управления ресурсами и создания презентаций

Рис. 6.2. *Значимость комплектов рабочих продуктов на протяжении жизненного цикла*

2. Требования: инструменты для управления требованиями
3. Проектирование: инструменты для визуального моделирования
4. Реализация: инструменты для компиляции/отладки, анализа кода, тестового покрытия и инструменты для управления тестированием
5. Внедрение: инструменты для анализа тестового покрытия и автоматизации тестирования, средства управления сетями, коммерческие компоненты (операционные системы, GUI, СУБД, сети, промежуточное ПО), средства инсталляции

Распределение ответственности среди групп, работающих над проектом, соответствует рабочим процессам, представленным в главе 8.

Комплект реализации в сравнении с комплектом внедрения

Отделение комплекта реализации (исходный код) от комплекта внедрения (исполняемый код) важно, поскольку для каждого из комплектов используются совершенно разные подходы. Структура информации, поставляемой пользователю (и, как правило, организации, выполняющей тестирование), сильно отличается от информационной структуры исходного кода. Решения при разработке, которые оказывают влияние на качество комплекта внедрения, но имеют относительно мало смысла для комплектов проектирования и реализации, включают в себя следующее:

- Динамически переопределяемые параметры (размеры буферов, цветовые палитры, число серверов, число одновременно работающих клиентов, файлы данных, параметры периода выполнения)
- Эффекты от оптимизации при компиляции/редактировании связей (такие, как оптимизация использования памяти вместо оптимизации быстродействия)
- Производительность при конкретной стратегии размещения (централизованная против распределенной, главные и вторичные процессы, динамическое распределение загрузки, немедленное резервное копирование против контрольных точек/отката)
- Ограничения виртуальной машины (описатели файлов, сборка мусора, размер динамически выделяемой памяти, максимальный размер записи, замена дисковых файлов)
- Параллелизм на уровне процессов (тупиковые ситуации и борьба за ресурсы)
- Различия в производительности или поведении в зависимости от платформы

Это важные исходные данные, которые должны содержаться либо в комплекте реализации (если эта информация встроена в исходный код), либо в комплекте внедрения (если она встроена в файлы с данными, файлы с параметрами, сценарии инсталляции или другие компоненты, ориентированные на конкретную среду). В динамически перенастраиваемых системах или переносимых компонентах представляется более удобным

отделять то, что относится к реализации в виде исходного кода, от того, что имеет отношение к среде выполнения (по причинам производительности, динамической адаптируемости или управления внесением изменений в исходный код). Такой подход позволяет абстрагироваться от конкретного типа платформы и от размеров и топологии используемой вычислительной инфраструктуры, которая включает в себя операционные системы, промежуточное ПО, сети и СУБД.

В качестве примера рассмотрим архитектуру ПО объемом в один миллион строк исходного кода. Система предназначена для оповещения о ракетных запусках (этот проект подробно описан в качестве практического примера в приложении D), к ней предъявляются чрезвычайно жесткие требования в отношении устойчивости к ошибкам и скорости обработки данных. В проекте могут создаваться разные конфигурации работающей системы из одного и того же исходного набора.

- Версия, включающая в себя только главный процесс обработки на основной машине для выполнения некоторого подмножества сценариев тестирования.
- Версия, включающая в себя главный и дублирующий процессы обработки на основной машине и допускающая в дальнейшем выполнение некоторых сценариев логической перенастройки.
- Версии, функционально эквивалентные двум предыдущим конфигурациям. Могут выполняться на предназначенных для них процессорах для оценки требуемой пропускной способности и времени реакции критичных сценариев в предполагаемой окончательной конфигурации.
- Версия, в которой главный серверный процесс может выполняться на одном процессоре, дублирующий серверный процесс — на другом, независимом процессоре, процесс тестирования/проверки на любом из них, а набор независимых клиентов с пользовательским интерфейсом — на пользовательских рабочих станциях. Последнее, на самом деле, и является конечной целевой конфигурацией, которая позволяет поддерживать диапазон динамических перенастроек.

Внедрение коммерческих продуктов у заказчиков может также охватывать широкий диапазон тестовых и рабочих конфигураций. Например, продукты класса промежуточного ПО являются быстродействующими надежными посредниками при запросе объектов (object request brokers), которые поставляются в реализациях для различных платформ, включая операционные системы рабочих станций, «голые» бортовые процессоры, операционные системы больших машин и различные операционные системы реального времени. Конфигурации продукта поддерживают различные компиляторы и языки, а также различные реализации сетевого ПО. Гетерогенный характер возможных конфигураций приводит к необходимости иметь чрезвычайно изощренную структуру исходного кода и гигантский набор различных рабочих продуктов внедрения.

6.1.3 Эволюция рабочих продуктов в течение жизненного цикла

Каждая стадия разработки вносит определенную точность в окончательное описание системы. В начале жизненного цикла точность мала, а представление слишком общее. В конце концов, точность представления становится высокой, и все рабочие продукты уточняются во всех деталях. В каждой точке жизненного цикла комплекты рабочих продуктов имеют различную степень завершенности. Однако все они должны находиться на сравнимых уровнях детализации и быть сопоставимы друг с другом в достаточной степени. Выполнение подробного анализа сопоставимости и непротиворечивости на ранних стадиях жизненного цикла (когда точность низка, а изменения вносятся часто) обычно приводит к низкой отдаче от инвестиций. По мере продвижения разработки архитектура стабилизируется, и подержание соответствия между различными рабочими продуктами начинает оправдывать затрачиваемые на него усилия.

В течение каждой стадии разработки особое внимание уделяется конкретному комплекту рабочих продуктов. По окончании стадии общее состояние системы прогрессирует за счет всех комплектов, как показано на рис. 6.3.

В начальной стадии основное внимание уделяется критичным требованиям, второстепенное внимание — начальным представлениям о внедрении, очень незначительное внимание — реализации, за исключением, быть может, выбора языка программирования и коммерческих компонентов. На самом верхнем уровне, возможно, некоторое внимание уделяется проектированию архитектуры, но не ее деталям.

На стадии уточнения удается достигнуть большей глубины проработки требований, большей полноты в комплекте проектирования. Вместе с тем продолжается работа над такими проблемами реализации и внедрения, как соглашения, касающиеся выполнения основных сценариев, и

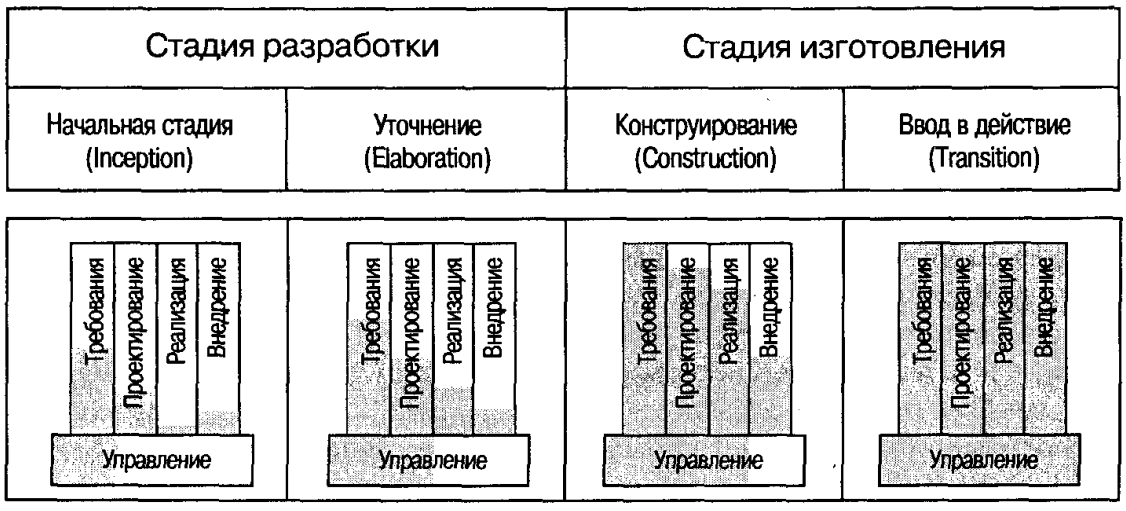


Рис. 6.3. *Эволюция комплектов рабочих продуктов в течение жизненного цикла*

анализ на предмет создания/покупки. Стадия уточнения включает в себя также создание работающего прототипа. Такой прототип вносит вклад во все четыре комплекта и позволяет точно оценить, являются ли интерфейсы и взаимодействие между компонентами непротиворечивыми и полными в контексте основных требований к системе и сценариев выполнения. На этом этапе, вообще говоря, присутствует широкое понимание интерфейсов компонентов, однако компоненты, создаваемые на заказ, проработаны пока не слишком глубоко. (Хотя коммерческие или иные существующие компоненты могут быть разработаны полностью.) Вклад, который вносится каждым из четырех комплектов, должен быть доведен до определенного уровня завершенности, чтобы могла быть создана базовая архитектура. Подобная эволюция требует соответствующей оценки рабочих продуктов из комплекта проектирования, комплекта реализации и комплекта внедрения на предмет соответствия критичным вариантам использования из комплекта требований, чтобы можно было с уверенностью предположить, что проект будет выполняться предсказуемо, а все возможные риски хорошо известны.

Главным на стадии конструирования являются проектирование и реализация. В начале этой стадии основное внимание должно обращаться на детальность рабочих продуктов проектирования. На более поздних этапах следует особо сосредоточиться на реализации разработки в виде исходного кода и отдельно тестируемых компонентов. Эта стадия должна приводить к почти окончательному наполнению комплектов требований, проектирования и реализации. Прорабатывается также комплект внедрения, по крайней мере, это касается тестирования одного или нескольких вариантов создаваемой системы посредством механизма альфа- или бета-версий.

Основное внимание на стадии ввода в действие уделяется достижению непротиворечивости и завершенности комплекта внедрения в контексте остальных комплектов. Устраняются оставшиеся дефекты, учитывается информация, полученная в результате альфа-, бета- и системного тестирования.

По мере продвижения разработки каждая из частей подвергается дальнейшему уточнению. Когда система готова, все четыре комплекта являются полностью завершенными и согласованными друг с другом. В отличие от традиционной практики не приходится сначала определять требования, затем выполнять проектирование и т.д. Напротив, развивается вся система целиком; решения, относящиеся к внедрению, могут оказывать влияние на требования, и это не означает выполнения всех работ с самого сначала. Ключевым является отказ от традиционного шаблона, когда по умолчанию считается, что один комплект предшествует другому. Напротив, смена одного состояния системы как целого другим, более совершенным состоянием системы сопровождается обычно эволюцией каждой из ее частей. На протяжении стадии ввода в действие соответствие комплекта требований и комплекта внедрения чрезвычайно важно. Изменяющийся комплект требований содержит сформировавшееся и точное представление заинтересованных сторон о критериях приемки, а комплект внедрения включает в себя реальный передаваемый

пользователю продукт. Следовательно, в течение переходного периода важной оказывается завершенность и отсутствие противоречий между двумя этими комплектами. Соответствие между другими комплектами является необходимым лишь постольку, поскольку это способствует разработке или управлению.

6.1.4 Рабочие продукты, связанные с тестированием

При традиционном тестировании ПО следуют подходу, в котором определяющей является документация. Команды разработчиков составляют документацию по требованиям, проектную документацию верхнего уровня и детальную проектную документацию до того, как начинают создаваться какие-либо исходные или выполняемые файлы. Аналогично, команды, проводящие тестирование, составляют документацию по планам тестирования системы, процедуре тестирования, планам интеграционного тестирования, планам тестирования модулей и процедуре тестирования модулей до начала создания каких-либо тестирующих драйверов, заглушек или инструментов. Для такого подхода с упреждающим созданием документации характерны те же проблемы при выполнении тестирования, что и при проведении разработки.

Одним из действительно определяющих принципов современного процесса является применение тех же комплектов, нотаций и рабочих продуктов для продуктов тестирования, которые используются при разработке самого продукта. Фактически мы лишь определяем инфраструктуру, необходимую для выполнения тестирования, как одно из обязательных подмножеств конечного продукта. Поступая таким образом, мы задействуем в процессе тестирования некоторые правила, присущие периоду разработки.

- Рабочие продукты тестирования должны создаваться параллельно с продуктом от начала до внедрения. Таким образом, тестирование — это деятельность, присущая всему жизненному циклу, а не только его поздним этапам.
- Рабочие продукты тестирования согласовываются и создаются в рамках тех же комплектов, что и сам продукт.
- Рабочие продукты тестирования реализуются в программируемом и воспроизводимом форматах (как и ПО).
- Рабочие продукты тестирования документируются тем же образом, что и сам продукт.
- Разработчики тестов используют те же инструменты, методы и процесс обучения, что и разработчики ПО, создающие основной продукт.

Эти правила позволяют добиваться значительной однородности в рамках рабочих процессов (см. главу 8). Каждый использует нотации и методы, присущие четырем комплектам рабочих продуктов, а не отдельным документам по разработке и тестированию. Взаимодействие между

отдельными разработчиками и анализ хода разработки для заинтересованных сторон могут осуществляться посредством меньшего числа форматов, меньшего числа специализированных нотаций, с меньшей неопределенностью и с большей эффективностью.

Тестирование — только один из аспектов процесса, связанного с оценкой. Другие аспекты — это инспектирование, анализ и демонстрация. Тестирование имеет дело с явными оценками при выполнении компонентов комплекта внедрения по контролируемому сценарию при наличии ожидаемых и реальных результатов. Успешность тестирования может быть определена сравнением ожидаемых результатов с реальными результатами с четко определенной математической точностью. На самом деле тестирование — это автоматизированная форма оценки.

Хотя подмножества рабочих продуктов тестирования весьма специфичны для каждого проекта, следующий пример показывает связи между рабочими продуктами тестирования и остальными комплектами. Рассмотрим проект, задачей которого является обработка сейсмических данных с целью поиска нефти. Система состоит из трех основных подсистем: (1) подсистема с датчиками, которые регистрируют непосредственные сейсмические данные в реальном времени и поставляют эти данные в (2) систему технической обработки данных; она преобразует исходные данные в организованную базу данных и управляет очередями запросов, поступающих от (3) терминальной подсистемы, позволяющей операторам рабочих станций изучать сейсмические данные в форме, удобной для человека. Для такой системы понадобятся следующие рабочие продукты тестирования:

- Комплект управления. Спецификации версии и определения версии содержат цели, критерии оценки и результаты промежуточной контрольной точки. Эти рабочие продукты состоят из планов тестирования и результатов тестирования, согласованных между проектными командами. Заявки на внесение изменений отталкиваются от результатов тестирования (дефекты, изменения пригодности к тестированию, двусмысленные требования, расширения) и критериев замыкания, связанных с внесением дискретных изменений в базовую структуру.
- Комплект требований. Системные варианты использования связаны с вопросами функционирования системы и описаниями приемочных тестов, включая ожидаемое поведение системы и ее характеристики качества. Полный комплект требований является рабочим продуктом тестирования, поскольку он служит основой для выполнения всех оценок в течение жизненного цикла.
- Комплект проектирования. В комплекте проектирования содержится модель тестирования компонентов, не предназначенных для внедрения. Эта модель необходима для принципиальной проверки продукта. Среди компонентов, входящих в комплект проектирования, могут быть эмулятор сейсмических событий для генерации

данных, похожих на данные, получаемые с настоящих датчиков; «виртуальный оператор», который может поддерживать проведение различных многочасовых тестов без присутствия человека; специальные инструменты для ранних демонстраций использования ресурсов; а также драйверы для проведения тестов по отдельным вариантам использования и драйверы для независимого тестирования отдельных компонентов.

- Комплект реализации. Самодокументируемые представления тестовых компонентов и тестовых драйверов в виде исходного кода являются эквивалентом тестовых процедур и тестовых сценариев. Среди исходных файлов могут быть и файлы с данными, которые может прочесть человек. В них хранятся статически определенные наборы данных, являющиеся непосредственными исходными файлами для тестирования. Выходные файлы тестовых драйверов являются эквивалентом отчетов о тестировании.
- Комплект внедрения. В него включаются исполняемые версии тестовых компонентов, тестовых драйверов и файлы с данными.

Для любой версии все рабочие продукты тестирования и готовые продукты поддерживаются с использованием одного и того же идентификатора основной версии. Они создаются, претерпевают изменения и устаревают как единое целое. Поскольку рабочие продукты тестирования формируются с применением тех же нотаций, методов и инструментария, подход к тестированию не противоречит подходу к проектированию и разработке. Такой подход заставляет поддерживать изменяющиеся рабочие продукты тестирования в таком виде, чтобы процесс регрессионного тестирования легко поддавался автоматизации.

6.2 РАБОЧИЕ ПРОДУКТЫ УПРАВЛЕНИЯ

Комплект управления включает в себя рабочие продукты, в которых содержатся промежуточные результаты и вспомогательная информация, необходимые для документирования унаследованных (legacy) продуктов/процессов, сопровождения продукта, улучшения продукта и усовершенствования процесса. Эти рабочие продукты подробно обсуждаются в последующих главах, посвященных детальному рассмотрению рабочих процессов и действий. Хотя в определениях для описания конкретных рабочих продуктов используется слово «документ», это означает всего лишь тот факт, что эти данные *могут быть* оформлены в виде бумажного документа. Во многих случаях данные могут обрабатываться, анализироваться и ими можно обмениваться исключительно в электронном виде.

Декомпозиция работ

Декомпозиция работ (work breakdown structure, WBS) является средством для определения и получения бюджета. Для того чтобы наблюдать и контролировать финансовое выполнение проекта, менеджер проекта

должен вникать в определение стоимости различных элементов проекта и в то, как эти суммы расходуются. Структура ответственности за расходование средств является серьезным ограничением при планировании проекта. Уроки, извлеченные из всех проектов, которые не были успешными, показали, что если WBS построена неправильно, она может увести процесс разработки и структуру продукта в неправильном направлении. Менеджеру проекта не следует расписывать более низкие уровни WBS (тем самым задавая определенные границы ответственности) до тех пор, пока не будет достигнут соответствующий уровень стабильности структуры продукта. Функциональное разбиение в WBS приведет к функциональной декомпозиции ПО. Концепция постоянно развивающейся WBS рассматривается в главе 10.

Бизнес-план

Рабочие продукты бизнес-планирования содержат всю информацию, необходимую для выяснения того, стоит ли инвестировать данный проект. Бизнес-план позволяет детализировать ожидаемый доход, ожидаемые затраты, технический и управленческий планы и дополнительные данные, необходимые для демонстрации рискованности и реализма планов. При обсуждении больших контрактов бизнес-план может быть реализован в виде полномасштабного предложения, содержащего большие объемы информации. В случае небольшого соглашения о коммерческом продукте он может быть реализован в виде краткого плана, к которому прилагается развернутый план. Основной задачей является превращение общей концепции системы в экономические термины с тем, чтобы организация могла произвести точную оценку ROI. Финансовые прогнозы постоянно эволюционируют, они заменяются более точными прогнозами по мере развития жизненного цикла. На рис. 6.4 приведена стандартная схема бизнес-плана.

I. Контекст (предметная область, рынок, область действия)

II. Технический подход

- A. План достижения набора функциональных возможностей
- B. План достижения требуемого качества
- C. Рабочие соглашения и технические риски

III. Подход к управлению

- A. График работ и оценка его рискованности
- B. Объективная мера успешного завершения проекта

IV. Дополнительные материалы

- A. Финансовый прогноз
  1. Оценка стоимости
  2. Оценка прибыли
  3. Основания для оценок

Рис. 6.4. *Типичная схема бизнес-плана*

Спецификации версии

Область действия, план и объективные критерии оценки для каждой базовой версии вытекают из концепции системы и из многих других источников (решение о приобретении/разработке, подходы к управлению рисками, архитектурные решения, смутные догадки, ограничения реализации, пороговые значения качества). Эти рабочие продукты развиваются вместе с процессом, становясь все более точными по мере развития жизненного цикла и более глубокого понимания требований. На рис. 6.5 приведена стандартная схема спецификаций версии.

Существуют две различные формы требований. Первая — общая концепция (или пользовательские потребности), на основе которой заключается соглашение между группой разработчиков и заказчиком. Эта информация будет изменяться, но медленно, по мере развития жизненного цикла. Она должна быть представлена в виде, понятном заказчику (специализированный формат, который может включать в себя текст, модели, варианты использования, электронные таблицы или другие форматы). Модель вариантов использования в контексте общей концепции служит для описания понятий, связанных с функционированием системы, в терминах, понятных пользователю/заказчику.

Второй формой требований, содержащихся в спецификациях версии, являются критерии оценки. Это временные цели для данной промежуточной контрольной точки жизненного цикла. Критерии оценки в спецификациях версии определяются как рабочие продукты управления, а не как часть комплекта требований. Они вытекают из общей концепции и многих других источников (решение о приобретении/разработке, подходы к управлению рисками, архитектурные решения, смутные догадки, ограничения реализации, пороговые значения качества). Требования, связанные с управлением, могут быть представлены в виде вариантов использования, реализаций вариантов использования, аннотаций к вариантам использования или в виде структурированного текста.

I. Содержание итерации

II. Измеримые цели

A. Критерии оценки

B. Степень завершенности

III. Демонстрационный план

A. График работ

B. Ответственность исполнителей

IV. Сценарии функционирования (демонстрируемые варианты использования)

A. Демонстрационные процедуры

B. Связи с концепцией и бизнес-планом

Рис. 6.5. *Типичная схема спецификаций версии*

Системные требования (касающиеся пользователя/заказчика) содержатся в общей концепции. Более низкие уровни требований определяются процессом (организованным итерационно, а не путем создания компонентов более низкого уровня) в виде критериев оценки (обычно задаваемых в виде вариантов использования и других текстуально представленных целей). Это означает, что требования более низкого уровня могут изменяться. Приведем пример:

1. Итерации начальной стадии. Обычно от 10 до 20 критериев определяют основные моменты, связанные с критичными вариантами использования, которые оказывают влияние на выбор альтернатив, касающихся архитектуры, и на общий бизнес-план.
2. Итерации стадии уточнения. Эти критерии оценки (возможно, около 50), применяемые к одной из возможных архитектур, определяют, что критичные варианты использования и критичные требования общей концепции могут быть сведены воедино с низким риском.
3. Итерации стадии конструирования. Эти критерии оценки (возможно, сотни) связаны с наиболее значимыми вариантами использования. То, что прошло оценку по этим критериям, образует полезные части продукта, которые можно превратить в формальный тест, в альфа- или бета-версии.
4. Итерации стадии ввода в действие. Полное множество вариантов использования и связанные с ними критерии оценки (возможно, тысячи) образуют критерии приемочного тестирования, связанного с внедрением некоторой версии в эксплуатацию.

Такой процесс естественно является эволюционным и легко соотносится с современной разработкой и изменяемой архитектурой. В конце концов, становится важной 100%-ная трассируемость требований, но промежуточные действия и этапы требуют непротиворечивости и завершенности в гораздо меньшей степени, чем в случае применения традиционного подхода. Критерии оценки каждой итерации отбрасываются, когда контрольная точка пройдена; это временные рабочие продукты. На каждом этапе создается более совершенная версия, поэтому содержание и извлеченные уроки сохраняются в каждом оказавшемся удачном наборе критериев оценки. Спецификации версий и соответствующие критерии оценки более важны на ранних этапах, так как позволяют убедиться в том, что разрешены проблемы с наиболее высокой степенью риска.

План разработки ПО

План разработки ПО превращает схему процесса в максимально детализированный план. Это определяющий документ для всего процесса работы над проектом. Он должен соответствовать контракту (если таковой имеется) и стандартам организации (при их наличии), изменяться параллельно с изменением проекта и требований и использоваться

I. **Контекст (область действия, цели)**

II. **Процесс разработки ПО**

- A. Составляющие проекта
  1. Стадии жизненного цикла
  2. Рабочие продукты
  3. Рабочие процессы
  4. Контрольные точки
- B. Область действия и содержимое основных контрольных точек
- C. Процедуры усовершенствования процесса

III. **Среда разработки ПО**

- A. Автоматизация процесса (конфигурация аппаратных и программных ресурсов)
- B. Процедуры распределения ресурсов (разделение ресурсов между организациями, безопасность доступа)

IV. **Управление внесением изменений в ПО**

- A. Общий план и процедуры управления конфигурацией
- B. Определения и процедуры выполнения запросов на внесение изменений в ПО
- C. Определения и процедуры базовой конфигурации

V. **Оценка ПО**

- A. Набор метрик и процедуры отчетности
- B. Процедуры управления рисками (идентификация, отслеживание и разрешение рисков)
- C. План оценки состояния проекта
- D. План приемочного тестирования

VI. ***Стандарты и процедуры***

- A. Стандарты и процедуры для рабочих продуктов

VII. **Дополнительные материалы**

- A. Область действия и содержание второстепенных контрольных точек
- B. Людские ресурсы (организация, план найма персонала, план обучения)

Рис. 6.6. *Типичная схема плана разработки ПО*

непротиворечивым образом всеми нижестоящими организациями, занятыми созданием ПО. Существуют два признака полезного плана: периодические изменения (он не является чем-то закоснелым), а также понимание и принятие его в равной степени как менеджерами, так и практиками. На рис. 6.6 приведена стандартная схема плана разработки ПО.

Описания версии

В документах, описывающих версию, приводятся результаты по каждой версии, включая оценку работоспособности по каждому из критериев оценки, содержащихся в соответствующих спецификациях версии. Версия должна сопровождаться документом, в котором описываются критерии оценки для данной конфигурации и который позволяет получить свидетельство (посредством демонстрации, тестирования, просмотра или анализа) того, что проверка по каждому критерию проводилась допустимым способом. Этот документ должен содержать также общую сводку значений параметров, которая бы давала количественную характеристику качеству данной версии в абсолютных и относительных терминах (по сравнению с предыдущими версиями, если таковые существуют). Здесь должны храниться документированные результаты «посмертного» рассмотрения любой версии, в том числе неразрешенные проблемы, рекомендации по улучшению процесса и продукта, соглашения об использовании критериев оценки, последующие действия и тому подобная информация. На рис. 6.7 приведена стандартная схема спецификаций версии.

I. Контекст
- A. Основное содержание версии
- B. Параметры версии

II. Замечания по версии
- A. Специфические для данной версии ограничения

III. Результаты выполнения оценок
- A. Обоснование выполненных оценок
- B. План последующих действий по неудовлетворенным критериям
- C. Рекомендации по следующей версии

IV. Неразрешенные проблемы
- A. Необходимые действия
- B. «Посмертное» заключение об извлеченных уроках

Рис. 6.7. *Типичная схема описания версии*

Оценки состояния

Оценки состояния предполагают периодическое создание «моментальных снимков» состояния проекта, которые включают в себя оценку риска менеджером проекта, показатели качества и показатели управления. Период между этими оценками может изменяться, но их необходимость сохраняется. Первостепенная задача хорошего процесса управления — убедиться в том, что ожидания всех заинтересованных сторон (подрядчика, заказчика, пользователя, субподрядчика) соответствуют друг другу и непротиворечивы. Документация по периодической оценке состояния является важным механизмом для управления потребностями всех

заинтересованных сторон на протяжении всего жизненного цикла; для выявления проблем управления, технических проблем и рисков проекта, для работы с ними и их разрешения; для ведения истории проекта. Они позволяют менеджеру держать руку на пульсе (см. главу 9).

Типичные оценки состояния должны включать в себя анализ ресурсов, укомплектованность кадрами, финансовые данные (затраты и прибыли), 10 самых серьезных рисков, технический прогресс («моментальные снимки» параметров), планы и результаты по основным контрольным точкам, общее состояние проекта или продукта, список действий и их последовательность. Постоянная открытая работа с объективными данными, полученными непосредственно из текущей деятельности и конфигурации изменяющегося продукта, обязательна для любого проекта.

База данных запросов на изменение ПО

Управление изменениями является одной из основных составляющих итерационного процесса разработки. Чем больше свобода внесения изменений, тем продуктивнее итерации проекта. Такая гибкость позволяет увеличивать объем выполненной работы, качество и число итераций, которые достижимы в рамках данного графика. На практике свобода внесения изменений достигается за счет автоматизации, а все бремя по управлению изменениями ложится на современную среду итерационного процесса разработки. Организационно процессы, зависящие от ручных методов управления изменениями, неэффективны. Следовательно, данные, необходимые для управления изменениями, преобразуются в важнейшие рабочие продукты управления, которые описываются как база данных для того, чтобы исподволь внушить понятие о необходимости автоматизации. Поскольку ПО создается на управляемой основе, все изменения должны формально отслеживаться и контролироваться. За счет автоматизации ввода данных и поддержки записей об изменениях в онлайновом режиме может быть автоматизирована большая часть бюрократической деятельности по управлению изменениями, сбору метрик и составлению отчетов. Запросы на внесение изменений подробно обсуждаются в главе 12.

Внедрение

Документ по внедрению может принимать множество форм. В зависимости от проекта, в него может входить несколько подмножеств документов для перевода продукта в рабочее состояние. В проектах с крупными контрактами, согласно которым система поставляется в другую эксплуатирующую организацию, отвечающую за сопровождение, рабочие продукты внедрения могут включать в себя рабочие руководства по компьютерной системе, руководства по инсталляции системы, планы и процедуры для перехода в новую среду эксплуатации (из существующей среды), описание вычислительной системы и т.п. Для коммерческих программных продуктов рабочие продукты внедрения могут содержать маркетинговые планы, средства запуска в продажу и курсы обучения.

Среда

Для современного подхода важно определить среду разработки и сопровождения как первоочередные рабочие продукты процесса. Стабильная интегрированная среда разработки должна поддерживать автоматизацию процесса разработки. Эта среда должна включать в себя управление требованиями, визуальное моделирование, автоматизацию ведения документации, средства программирования для клиента и сервера, автоматизированное регрессионное тестирование, интегрированное управление изменениями и отслеживание дефектов. Общим для успешных проектов

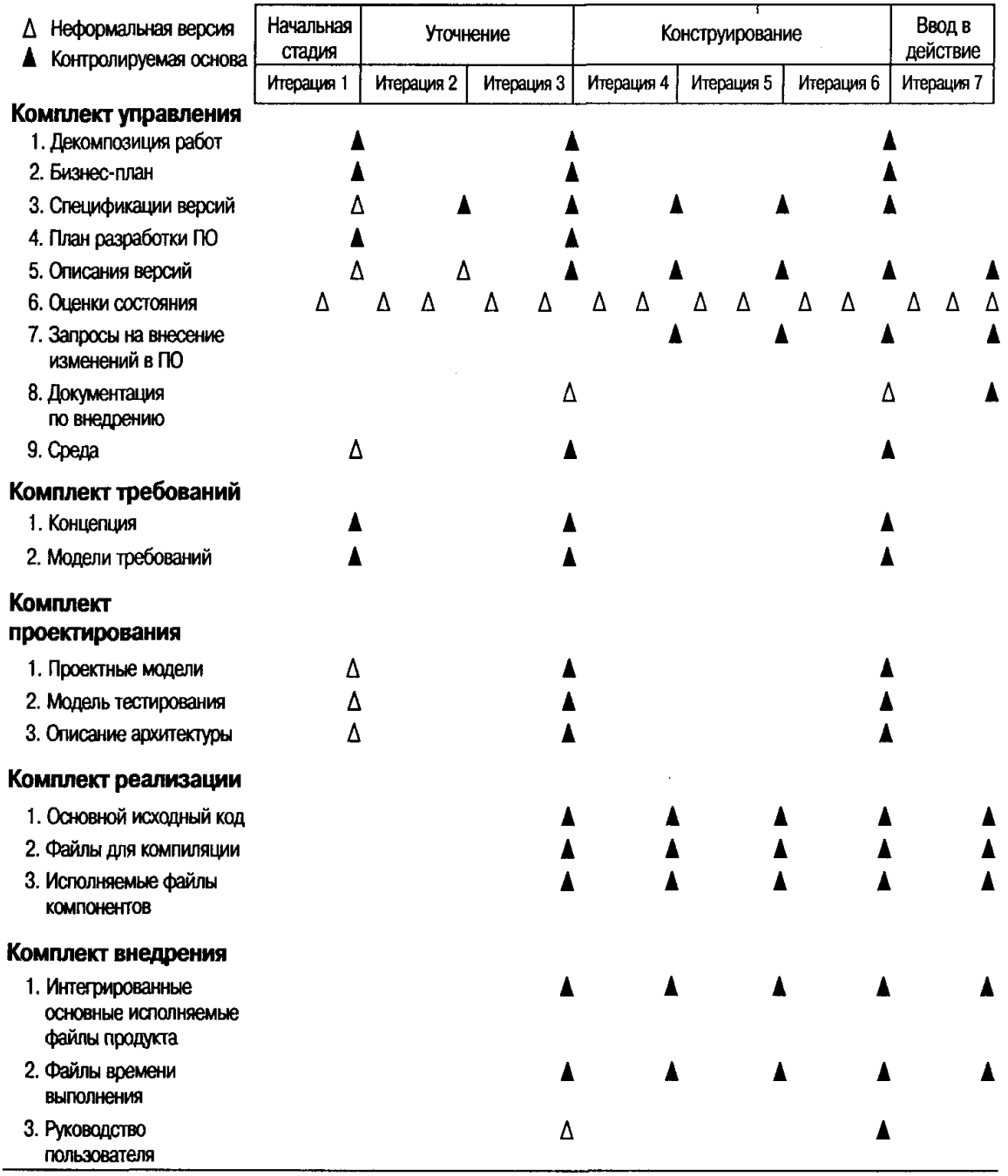


Рис. 6.8. *Последовательности рабочих продуктов на протяжении типичного жизненного цикла*

по созданию ПО является то, что для их выполнения нанимают квалифицированных специалистов и обеспечивают их хорошими инструментами. Автоматизация процесса разработки ПО дает выигрыш в качестве, в возможности оценивать затраты и сроки и в общей производительности при использовании меньшей команды. Позволяя разработчикам быстро ориентироваться в рабочих продуктах разработки и с легкостью поддерживать их в актуальном состоянии, интегрированные наборы инструментов играют все возрастающую роль при пошаговой и итерационной разработке.

Последовательности рабочих продуктов управления

На каждой стадии жизненного цикла создаются новые и обновляются ранее созданные рабочие продукты. Они учитывают накопленный в проекте опыт и уточняют принимаемые решения. Некоторые рабочие продукты обновляются в каждой основной контрольной точке, другие – в каждой второстепенной. На рис. 6.8 показаны типичные последовательности рабочих продуктов на разных стадиях жизненного цикла.

6.3 РАБОЧИЕ ПРОДУКТЫ РАЗРАБОТКИ

Большинство рабочих продуктов разработки представлено в строгих нотациях, таких как UML, языки программирования или машинные коды. Поскольку эта книга написана с точки зрения управления проектом, мы не будем подробно останавливаться на этих рабочих продуктах. Однако три вида продуктов разработки заслуживают уточнения.

Общая концепция

Данный документ содержит полное общее представление относительно разрабатываемой программной системы и обеспечивает основу для заключения контракта между ответственным за финансирование и организацией-разработчиком. Независимо от того, является ли проект огромной разработкой, проводимой по военным стандартам (общая концепция которого может представлять собой 300-страничные системные спецификации), или маленьким, финансируемым внутри организации коммерческим продуктом (чья концепция может занимать две странички), у каждого проекта должен быть источник, в котором бы содержались ожидания всех заинтересованных сторон. Подразумевается, что общая концепция проекта будет изменяться по мере изменения понимания требований, архитектуры, планов и технологии. Хороший документ с общей концепцией должен меняться медленно. На рис. 6.9 приводится стандартная общая схема концепции.

Концепция пишется с точки зрения пользователя, при этом основное внимание уделяется основным функциональным возможностям системы и достижению приемлемого уровня качества. Документ должен содержать по крайней мере два приложения. В первом приложении описываются аспекты функционирования будущей системы с применением вариантов использования (визуальная модель и отдельные рабочие продукты). Во втором приложении рассматриваются риски, которые могут

I. Описание набора функциональных возможностей
- A. Старшинство и приоритет

II. Показатели и диапазоны качества

III. Необходимые ограничения
- A. Внешние интерфейсы

IV. Дополнения
- A. Варианты использования
  1. Основные сценарии
  2. Критерии приемки и допустимые отклонения
- B. Желаемая гибкость (возможные сценарии внесения изменений)

Рис. 6.9. *Типичная схема документа концепции*

возникнуть при внесении изменений, чтобы можно было предпринять защитные меры в процессе проектирования.

Концепция должна содержать в себе как описание того, что будет включено в проект, так и описание тех возможностей, которые рассматривались, но не были включены. Должны указываться также эксплуатационные характеристики (объемные показатели, время реакции, точность), профили пользователей и интерфейсы взаимодействия с объектами, находящимися за границами системы (в тех случаях, когда это может понадобиться). Не следует определять концепцию только для начального эксплуатационного уровня; его наиболее вероятный путь развития должен быть отражен таким образом, чтобы в этом контексте могла быть оценена применимость разработки. Аспект функционирования включает в себя определение вариантов использования и сценариев для нормальных ситуаций и различных отклонений. Представление вариантов использования обеспечивает динамический контекст для понимания и уточнения области действия, для оценки целостности проектной модели и для разработки процедур приемочного тестирования. Документ с общей концепцией обеспечивает основу договоренностей о требованиях, понятных всем заинтересованным сторонам.

Описание архитектуры

Описание архитектуры — это организованное определенным образом представление архитектуры разрабатываемого ПО. Оно извлекается в основном из проектной модели и включает в себя представление рабочих продуктов проектирования, реализации и внедрения, которые необходимы для понимания аспектов функционирования системы, вытекающих из требований. Масштаб описания архитектуры будет меняться от проекта к проекту в зависимости от многих факторов. Архитектура может быть описана с использованием подмножества проектной модели, или как некая абстракция проектной модели с привлечением дополнительного материала, или как комбинация того и другого. В качестве примера этих двух форм описаний рассмотрим архитектуру настоящей книги.

I. **Общее представление архитектуры**
   A. Цели
   B. Ограничения
   C. Степени свободы

II. **Различные представления архитектуры**
   A. Проектное представление
   B. Представление процессов
   C. Представление компонентов
   D. Представление внедрения

III. **Архитектурные взаимодействия**
   A. Аспект функционирования для основных сценариев
   B. Аспект функционирования для дополнительных сценариев
   C. Аспект функционирования для аномальных условий

IV. **Производительность**

V. **Обоснования и соглашения**

Рис. 6.10. *Типичная схема описания архитектуры*

- Для формы описания в виде подмножества вполне достаточно оглавления. Такое описание архитектуры книги получается непосредственно из самой книги.
- Для абстрактной формы достаточно трактовки «Cliffs Notes». (Cliffs Notes (увлекательные заметки) — это сжатые версии классических книг, используемые в качестве учебников некоторыми учащимися колледжей.) Данный формат является абстракцией, которая создается независимо и включает в себя дополнительный материал, который не может быть получен непосредственно из изменяющегося продукта.

Подход, рассматриваемый в разделе 7.2, позволяет подогнать описание архитектуры под специфические требования конкретного проекта. На рис. 6.10 приведена стандартная схема описания архитектуры.

Руководство пользователя

Это руководство обеспечивает пользователя справочной документацией, необходимой для поддержки переданного в эксплуатацию ПО. Хотя его содержание может существенно меняться в зависимости от области применения, в руководство пользователя должны включаться как минимум процедуры инсталляции, процедуры и руководство по использованию, эксплуатационные ограничения и описание пользовательского интерфейса. Для программных продуктов, обладающих пользовательским интерфейсом, руководство следует создавать на ранних стадиях жизненного цикла, поскольку это оказывается необходимым механизмом для обсуждения и стабилизации некоторого важного подмножества требований. Руководство пользователя должно писаться членами команды, выполняющей тестирование; для них характерно более глубокое понимание

пользовательской точки зрения, чем для команды разработчиков. Если за создание руководства пользователя отвечает команда, выполняющая тестирование, оно может формироваться параллельно с разработкой и применяться на ранних стадиях в качестве существенного и уместного в перспективе критерия оценки. Кроме того, оно может представлять собой необходимую основу для планов тестирования и тестовых вариантов, а также для автоматизированного построения наборов тестов.

6.4 РАБОЧИЕ ПРОДУКТЫ, ИСПОЛЬЗУЕМЫЕ НА ПРАКТИКЕ

Традиционный подход, при котором разработка документации являлась определяющей, приводил к пустой трате невероятного количества рабочего времени на создание, редактирование, чтение, обновление и распространение документов. Почему? Есть несколько причин, обуславливающих важность таких документов для процесса. Во-первых, не существовало строгих методов разработки и языков для спецификации требований и проектирования. Следовательно, стандартным форматом являлся бумажный документ со специализированным текстом и графическими схемами. Во-вторых, традиционные языки, использовавшиеся на этапах реализации и внедрения, были малопонятны и плохо структурированы. Для того чтобы передать подробную структуру и поведение ПО другим заинтересованным сторонам (тем, кто тестирует, сопровождает, управляет), требовался формат, более пригодный для чтения человеком. Возможно, наиболее важной была необходимость «заслуживающей доверия» оценки достижений в области ПО. Документы являли собой осязаемый, но порождающий ошибки механизм демонстрации достижений.

В некоторых областях подходы с преобладающей разработкой документации за последние 30 лет превратились в основное препятствие на пути совершенствования процесса. Качество документов стало более важным, чем качество представляемых ими результатов. А оценка качества человеком, читающим абстрактные описания, — процесс весьма субъективный. Много усилий тратилось на оценку одномерных проблем, лежащих на поверхности, и мало внимания уделялось многомерным проблемам, которые определяют такие качества архитектуры, как производительность и адаптируемость.

Циклы создания документов, их анализа и обновления приводили к появлению зримых и формальных свидетельств движения по графику, создавая тем самым зависимость от графика работ и числа точек синхронизации. Например, следующий сценарий являлся не таким уж необычным для больших оборонных проектов: сначала тратился месяц на подготовку проектного документа, потом документ передавался заказчику и приходилось ждать еще месяц, чтобы получить комментарии к этому документу, после чего еще месяц тратился на написание ответа на эти комментарии и на внесение изменений. При наличии большого числа многомесячных циклов по работе с документами, которыми приходилось управлять и которые приходилось расписывать и синхронизировать, нет

ничего удивительного в том, что жизненный цикл разработки проектов растягивался на пять лет. Длинные и очень подробные документы, которые обычно воспринимались как средство демонстрации очередных достижений, приводили к преждевременному рассмотрению деталей работы и к увеличению числа отбраковок и переделок на более поздних стадиях жизненного цикла.

Более эффективным подходом является переориентация усилий, направленных на работу с документами, на увеличение строгости и понятности источников информации и на обеспечение просмотра в онлайновом режиме естественного источника информации посредством использования более совершенных поисковых и навигационных инструментов. Такой подход приводит к исключению из процесса огромного числа непродуктивных отбраковок и переделок и позволяет постоянно участвовать в работе всем тем, кто имеет непосредственное отношение к рабочим продуктам.

Эта философия приводит к возникновению следующих проблем:

- **Люди заинтересованы в анализе информации, но не понимают языка рабочих продуктов.** Многие из заинтересованных лиц будут сопротивляться необходимости изучения того профессионального языка, на котором изложены продукты. Нередко можно встретить людей (например, менеджеров-ветеранов, опытных специалистов в области качества или проверяющих из регулирующего органа), чья реакция будет следующей: «Я не собираюсь учить UML, но я хочу познакомиться с проектом этого ПО, поэтому подготовьте мне отдельное описание в виде диаграмм и текста, которое будет мне понятно». Будем ли мы реагировать на аналогичное требование того, кто изучает чертежи здания? Нет. Мы потребуем, чтобы тот, кто изучает чертежи, был знаком с инженерной системой обозначений. Точно так же мы должны прекратить опекать публику, которая отказывается рассматривать программирование как инженерную дисциплину. Такие заинтересованные стороны обычно приводят к повышению стоимости и сроков проекта, не увеличивая его ценности.

- **Люди хотят ознакомиться с информацией, но не имеют доступа к необходимым инструментам.** Нередко бывает так, что организация-разработчик не обеспечена всеми необходимыми инструментами; еще реже заинтересованные стороны имеют какую-либо возможность для знакомства с материалами в онлайновом режиме. Следовательно, организации вынуждены обмениваться бумажными документами. Стандартные форматы (такие, как UML, электронные таблицы, Visual Basic, C++ и Ada 95), инструментарий для визуализации и Web делают электронный обмен информации между всеми участниками экономически оправданным. Подход к рабочим продуктам — это единственная область, в которой оптимальный процесс разработки ПО может быть нарушен в том случае, если философия процесса принимается не всеми его участниками.

- **Рабочие продукты, предназначенные для просмотра человеком, должны использовать строгие нотации, которые являются полными, непротиворечивыми и самодокументируемыми.** Для всех идентификаторов и определений должны использоваться правильные английские слова. Акронимы и аббревиатуры следует применять только там, где этот жаргон является общепринятым в контексте использования компонентов. Независимо от того, какие применяются языки и инструменты, нет смысла сокращать или зашифровывать исходные идентификаторы языков программирования и моделирования. Уменьшение числа нажатий клавиш за счет использования сокращений может, конечно, упростить труд автора рабочих продуктов, но приведет к появлению ошибок на остальных стадиях жизненного цикла. Запрещение такой практики даст выигрыш и в производительности, и в качестве. ПО пишется только один раз, но читается много раз. Следовательно, необходимо уделять особое внимание удобочитаемости и требовать использования правильных английских слов во всех рабочих продуктах. Такая практика позволяет получать понятные представления, удобные для просмотра форматы («безбумажное» рассмотрение), нотации повышенной строгости и снижение количества ошибок.

- **Полезная документация самодостаточна: это документация, которая используется.** Кроме всего прочего, создание самодокументируемых рабочих продуктов дает организации-разработчику «право» работать исключительно со строгими нотациями и избегать создания отдельных документов для описания всех деталей модели, компонента или процедуры тестирования. Если обнаружится, что некоторая информация, в частности какой-либо документ, создается, но не используется, следует отказаться от нее в пользу той, что реально применяется для достижения желаемой цели. Старайтесь улучшить ее самодокументируемость.

- **Бумага осязаема; электронные материалы слишком легко изменить.** Одна из причин, по которой некоторые участники предпочитают бумажные документы, заключается в том, что после того, как бумаги получены, они становятся осязаемыми, постоянными и неизменными. Рабочие продукты, доступные в онлайновом режиме и в Web, могут быть легко изменены и потому рассматриваются с большим скептицизмом. Электронные материалы должны и будут приветствоваться многими участниками проекта, переход всего мира на такую работу — вопрос времени. Преимущества оказываются существенными и глубокими во многих областях. Все смогут убедиться, что инструментарий и среда будут развиваться в направлении поддержки управления изменениями, системных журналов, электронных подписей и других преимуществ групповой работы, так что электронный обмен информацией заменит бумажный.

Чрезвычайно важно, чтобы основное внимание уделялось информации, содержащейся в рабочих продуктах, а не бумаге. Краткие документы обычно более полезны, чем длинные. Главный продукт – это ПО; документация является всего лишь вспомогательным материалом.

Глава 7

Архитектура ПО, основанная на моделях

Ключевые моменты

- Архитектура — это проект программной системы.
- Единственной целью стадии разработки является получение устойчивой базовой архитектуры.
- Базовая архитектура — не бумажный документ; это некоторая совокупность информации, пронизывающая всю разработку системы.
- Архитектура описывается посредством извлечения существенной информации из проектных моделей.

Архитектура ПО является центральной проблемой при разработке сложных программных систем точно так же, как архитектура здания является центральной проблемой для небоскреба. Однако архитектура ПО имеет несколько дополнительных параметров сложности. В отличие от архитектуры большого здания, критичные параметры производительности и функциональные возможности сложной программной системы не могут быть описаны с помощью устоявшихся физических законов. Они не описываются и какими бы то ни было общепринятыми математическими формулами. Таким образом, у архитекторов ПО отсутствуют основные неопровержимые принципы. Существует много эвристических подходов и расплывчатых указаний, однако фундаментальные единицы измерения того, что такое хорошо, сильно зависят от конкретной ситуации. Поскольку устоявшейся теории нет, архитекторы ПО вынуждены полагаться на проведение всякого рода экспериментов при попытке сформулировать архитектуру ПО. Это является одной из причин перехода к итерационному процессу, при котором на ранних стадиях уделяется особое внимание и стимулируется эволюция архитектуры посредством создания прототипов и демонстраций.

В предыдущих главах содержится много рассуждений по поводу архитектуры, но не дано определение этого термина. В индустрии ПО общепринятое определение архитектуры также отсутствует. В настоящей главе собраны некоторые точки зрения на архитектуру и строится контекст, в котором могли бы стать понятными принципы управления процессом с упреждающей разработкой архитектуры.

Поскольку ранние программные системы были менее мощными, чем современные системы, их архитектуры были намного проще и для них было достаточно неформального представления. В системе, состоящей из одной программы, выполняющейся на одном компьютере, соответствие между объектами разработки, объектами реализации и объектами внедрения было тривиальным. В современных сложных программных системах нам приходится применять множество различных моделей и представлений для того, чтобы иметь возможность воспользоваться преимуществами современных технологий, таких как коммерческие компоненты, объектно-ориентированные методы, открытые системы, распределенные системы, среда разработки и среда выполнения, современные языки. *Модель* — это относительно независимое абстрактное представление системы. *Представление* (view) — это часть модели, которая позволяет описать какую-либо конкретную, относящуюся к делу точку зрения.

7.1 АРХИТЕКТУРА С ТОЧКИ ЗРЕНИЯ УПРАВЛЕНИЯ

Наиболее важным техническим продуктом проекта ПО является архитектура: инфраструктура, управление и интерфейсы данных, которые позволяют компонентам ПО функционировать в качестве единой системы, а разработчикам ПО эффективно трудиться в единой команде. Организация выверенного и точного взаимодействия между группами людей — это проблема на все времена для любой организации. Когда в средства взаимодействия входят множество языков и межгрупповые различия в грамотности, проблема взаимодействия может стать чрезвычайно сложной и даже неразрешимой. Для того чтобы команде разработчиков сопутствовал успех, взаимодействие в рамках проекта — как это отражено в архитектуре ПО — должно быть и выверенным, и точным.

С точки зрения управления существуют три различных аспекта архитектуры:

1. *Архитектура* (неосязаемое понятие проекта) — это разработка программной системы в отличие от разработки компонента. В нее входят все виды работ, необходимые для определения полной спецификации рабочих продуктов. Принимаются значимые решения по вопросам покупки/создания, и все компоненты, изготовляемые на заказ, усовершенствуются таким образом, чтобы можно было с уверенностью оценить затраты на каждый отдельный компонент и затраты на создание/сборку.

2. *Базовая архитектура* (осязаемые рабочие продукты) — это часть информации, распределенной по всем комплектам продуктов, касающимся разработки, достаточная, чтобы все заинтересованные стороны могли убедиться в том, что общая концепция системы (функции и качество) может быть реализована в рамках бизнес-плана (затраты, прибыль, время, технология, люди).

3. *Описание архитектуры* (представление архитектуры в виде, пригодном для человеческого восприятия, что является одним из компонентов базовой архитектуры) — это организованное подмножество информации, извлеченное из проектных моделей. В него входит дополнительная специальная нотация (текстовая и графическая), которая необходима для прояснения информации, содержащейся в моделях. Описание архитектуры устанавливает соответствие того, как неосязаемое понятие реализуется в осязаемых материалах.

Эти определения довольно абстрактны, поскольку архитектура принимает различные формы в разных системных областях. В частности, число различных представлений и уровень детализации каждого представления могут варьироваться в широких пределах. Архитектура краткой справки, например, имеет более простую форму, чем архитектура полнометражного художественного фильма, хотя оба продукта могут являться различными формами представления биографии. Архитектура планера имеет более простую форму, чем архитектура тяжелого реактивного лайнера, хотя оба продукта — летательные аппараты. Похожим образом архитектура ПО системы контроля за движением в воздушном пространстве существенно отличается от программной архитектуры какого-либо маленького инструмента, предназначенного для разработки.

Важность архитектуры ПО и ее тесная связь с современными процессами разработки ПО могут быть сформулированы следующим образом:

- Создание устойчивой архитектуры ПО — значимая контрольная точка проекта, когда должны быть приняты все важные решения о покупке/создании. Это событие в жизненном цикле представляет собой переход от стадии разработки проекта, характеризующейся большой степенью неопределенности, к стадии производства, характеризующейся управлением по предсказуемому плану разработки.

- Представление архитектуры является основой для достижения компромиссов между проблемной областью (требования и ограничения) и областью решений (рабочий продукт).

- Архитектура и процесс заключают в себе множество важных (с высокой степенью риска и компромиссов) взаимоотношений между людьми, командами, организациями и посредниками.

- Слабая архитектура и неразвитый процесс часто представляются как причины, по которым проект потерпел неудачу.

- Развитый процесс, понимание основных требований и наглядная архитектура являются важными предпосылками для предсказуемого планирования.
- Разработка архитектуры и описание процесса являются интеллектуальными шагами, которые приводят к решению проблемы без нарушения ограничений; они требуют творческого подхода и не могут быть автоматизированы.

7.2 АРХИТЕКТУРА С ТЕХНИЧЕСКОЙ ТОЧКИ ЗРЕНИЯ

Архитектура ПО обсуждается на протяжении всего последнего десятилетия, однако прийти к единым определениям, терминологии и принципам пока не удалось. Следующее ниже обсуждение развивает общие основы архитектуры, разработанные в корпорации Rational Software, и, в частности, концепции архитектуры ПО Филиппа Кручтена [Kruchten, 1995].

Архитектура ПО заключает в себе структуру программных систем (выбор элементов и их объединение в более крупные подсистемы), их поведение (взаимодействие между элементами) и образцы, которые управляют этими элементами, их взаимодействиями и их сочетанием. Контекст структуры, поведения и образцов ПО должен включать в себя функциональные возможности, производительность, устойчивость к внешним воздействиям, понятность, экономические соглашения, технологические ограничения и вопросы эстетики.

Архитектурная схема описывается в терминах представлений (views), которые являются абстракциями проектных UML-моделей. Проектная модель (design model) включает в себя всю полноту и глубину информации. Архитектурное представление – это абстракция проектной модели; в нем содержится только информация, значимая для архитектуры. Для большинства реальных систем требуются четыре различных представления: с точки зрения проекта, процессов, компонентов и внедрения. Их задачи могут быть описаны следующим образом:

- Проект: описывает архитектурно значимые структуры и функции, отраженные в проектной модели.
- Процесс: описывает взаимосвязь между параллелизмом и потоками управления, с одной стороны, и остальными представлениями, с другой стороны.
- Компоненты: описывает структуру рабочих продуктов, касающихся реализации.
- Внедрение: описывает структуру рабочих продуктов, касающихся внедрения.

Представление с точки зрения проекта, вероятно, необходимо для каждой системы; остальные три представления могут добавляться в случае сложных систем. Например, для каждой распределенной системы

потребуется представление с точки зрения процессов и представление с точки зрения внедрения. Большим системам, а также смешанным системам, состоящим из компонентов, изготовленных на заказ, и коммерческих компонентов, потребуется отдельное представление с точки зрения компонентов.

На рис. 7.1 показаны все рабочие продукты, входящие в проектный комплект, включая архитектурные представления и описание архитектуры. Описание архитектуры обычно хранится в электронном виде, но всегда поддерживается таким образом, что может быть распечатано в виде отдельного связного документа. Рабочие модели и архитектурные представления определяются как наборы UML-диаграмм.

Модель требований представляет поведение системы так, как это видится ее конечным пользователям, аналитикам и тестировщикам. Это представление моделируется статически с применением диаграмм вариантов использования и классов, а также динамически с применением диаграмм последовательности, состояния, деятельности и кооперативных диаграмм.

- *Представление вариантов* использования описывает, каким образом критичные для системы (архитектурно значимые) варианты использования реализуются с помощью элементов проектной модели. Оно моделируется как статически — с применением диаграмм вариантов использования, так и динамически — с помощью любых UML-диграмм поведения.

Проектная модель описывает архитектуру системы и разработку ее компонентов в рамках данной архитектуры, включая функциональную структуру, структуру параллельных процессов, структуру реализации и функционирования для области решения, так, как это видится ее разработчикам. Статические описания снабжаются структурными диаграммами (диаграммами классов, объектов, компонентов, размещения). Динамические описания снабжаются любыми UML-диаграммами поведения (кооперативными диаграммами, диаграммами последовательности, состояния и деятельности).

- *Проектное представление* описывает архитектурно значимые элементы проектной модели. Это представление является абстракцией проектной модели и касается основной структуры и функциональности системы. Оно моделируется статически с использованием диаграмм объектов и классов, а также динамически с применением любых UML-диаграмм поведения.
- *Представление процессов* связано с проблемами взаимодействия в период выполнения, касающимися реализации архитектуры с помощью распределенной модели среды эксплуатации, включая логическую топологию сети ПО (распределение процессов и потоков (threads) управления), межпроцессное взаимодействие и управление состоянием. Это представление моделируется как статически — с использованием диаграмм размещения, так и динамически — с применением любых UML-диаграмм поведения.

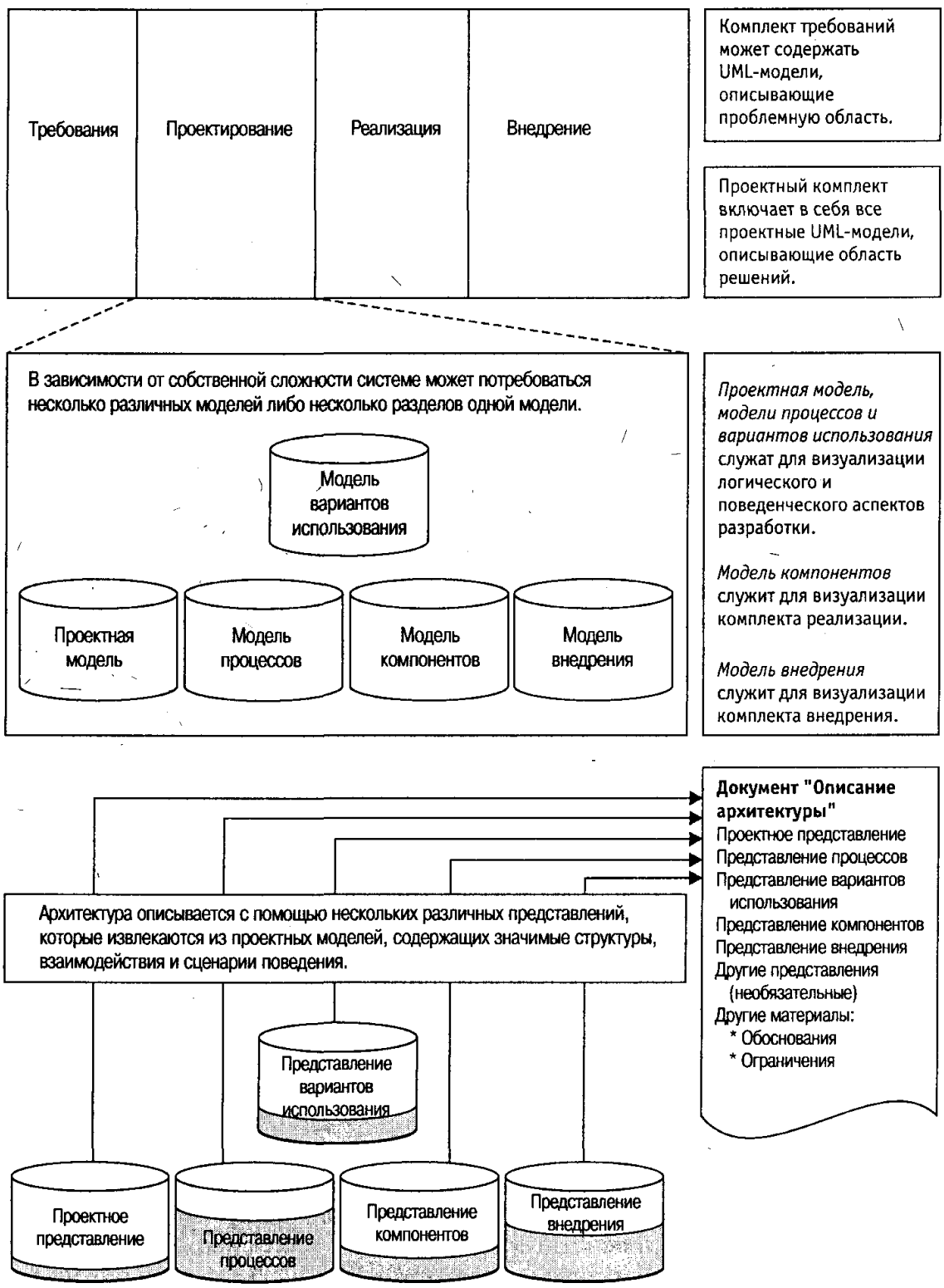


Рис. 7.1. *Архитектура: организованное и абстрактное представление проектных моделей*

- *Представление компонентов* описывает архитектурно значимые элементы комплекта реализации. Это представление является абстракцией проектной модели и касается реализации исходного кода ПО с точки зрения системных интеграторов и разработчиков, особенно с учетом версий и управления конфигурацией. Оно моделируется как статически — с использованием диаграмм компонентов, так и динамически — с применением любых UML-диаграмм поведения.
- *Представление внедрения* имеет отношение к исполняемой реализации системы, включая распределение логических процессов (логическая топология ПО) по физическим ресурсам сети в среде эксплуатации (физическая топология системы). Это представление моделируется как статически — с использованием диаграмм размещения, так и динамически — с применением любых UML-диаграмм поведения.

Описания архитектуры принимают различные формы и виды в разных организациях и областях знаний. В любой конкретный момент времени архитектуре требуются подмножества рабочих продуктов из всех комплектов, касающихся разработки. Реальный уровень содержимого в каждом комплекте зависит от конкретной ситуации, и существует очень немного хороших эвристических подходов для объективного описания того, что является архитектурно значимым, а что — нет.

Вообще говоря, базовая архитектура должна включать в себя следующее:

- Требования: критичные варианты использования, требуемое качество на уровне системы в целом и приоритетные связи между возможностями и качествами системы.
- Проект: наименования, атрибуты, структуры, поведение, группирование и связи значимых классов и компонентов.
- Реализация: исходное описание компонентов и спецификация (номер, название, цель, стоимость) всех составляющих компонентов.
- Внедрение: исполняемые компоненты, необходимые для демонстрации критичных вариантов использования, и риск, сопутствующий достижению качеств системы.

Хотя технические детали описания архитектуры не являются главными при управлении созданием ПО, дух, лежащий в основе упреждающей разработки архитектуры, крайне важен для достижения успеха. Проведение черты (отделяющей то, что имеет отношение к архитектуре, от того, что не имеет) является вызовом менеджерам проекта, поскольку такая черта определяет баланс, который оказывает значительное влияние на успех всего проекта.

Базовая архитектура описывается как сбалансированное подмножество информации из всех комплектов, в то время как описание архитектуры полностью содержится внутри проектного комплекта. Это является небольшим, но важным отличием между традиционными подходами и

современными итерационными процессами разработки. При традиционных подходах базовая архитектура приравнивается к описанию архитектуры (выполняемому в виде документа, который не имеет строгой нотации) без какого бы то ни было представления в других комплектах рабочих продуктов, касающихся разработки, для подтверждения целостности описания. При итерационном подходе базовая архитектура является частичной реализацией описания архитектуры, которое оказывается ощутимым свидетельством того, что данная архитектура является допустимой в контексте текущих требований и планов.

Описание архитектуры может принимать самые разнообразные формы, начиная от простого, непосредственного подмножества UML-диаграмм и заканчивая сложным набором моделей с широким диапазоном различных представлений, которые содержат в себе отдельные аспекты сложных систем. Первое может подойти для небольшой высокопрофессиональной команды, создающей некий инструмент для разработки, второе — для распределенной крупномасштабной системы управления и контроля, сбой которой способен привести к катастрофическим последствиям.

Комплекты рабочих продуктов развиваются на протяжении всего жизненного цикла: от стадии разработки (когда основное внимание уделяется требованиям и проекту) до стадии производства (когда центр внимания перемещается на реализацию и внедрение). Точка перехода от стадии разработки к стадии производства представляет собой состояние, при котором проект достигает стабильной базовой архитектуры. С точки зрения управления это состояние достигается тогда, когда соответствующие заинтересованные стороны соглашаются с тем, что общая концепция системы (определяемая требованиями и архитектурой, представленной в проектном комплекте и частично осуществленной в комплектах реализации и внедрения) может быть реализована с хорошо предсказуемыми затратами и сроками (так, как это подтверждается в комплекте управления). Доказательство этого утверждения обычно требует не только документов и проведения брифингов, но и наличия работающих прототипов, которые демонстрируют развивающиеся возможности системы. Обратная связь от таких демонстраций оказывает осязаемое влияние на обоснованность принимаемого решения. Чем больше используется стандартных компонентов, тем проще достигается это состояние. Чем больше применяется компонентов, изготовленных на заказ, тем сложнее достигается это состояние, и тем сложнее оценить затраты на создание.

Глава 8

Рабочие процессы

Ключевые моменты

- Организационно все виды работ подразделяются на семь основных рабочих процессов: управление, среда, требования, проектирование, реализация, оценка и внедрение.
- Эти работы выполняются параллельно, с изменяющимися по ходу жизненного цикла уровнями трудозатрат и уделяемого внимания.
- Процесс управления состоит в основном из трех дисциплин: планирование, контроль за ходом проекта и организация.

Большинство описаний процесса в качестве главного способа его представления использует последовательность выполнения работ. Описания процесса, ориентированные на последовательность выполняемых работ, несложно понимать, представлять, планировать и проводить в жизнь. С точки зрения отдельной личности все виды работ сугубо последовательны. Однако простые последовательности работ совершенно нереальны для проектов по созданию ПО, реализуемых командой. В эту работу может включаться несколько команд, создающих множество рабочих продуктов, которые необходимо синхронизировать, взаимно проверять, добиваться их однородности, соединять и интегрировать. Распределенный характер процесса по созданию ПО и зависящие друг от друга рабочие процессы являются основным источником сложности управления.

Один из самых тонких пороков традиционного процесса создания ПО заключался в представлении макропроцесса жизненного цикла в виде последовательно выполняемых работ: от анализа требований к проектированию, кодированию, тестированию и внедрению. Если говорить абстрактно, то в успешных процессах реализовывалась именно такая последовательность, однако границы между стадиями были размыты, и это принималось как данность неантагонистически настроенными заинтересованными сторонами. Неудачные проекты, с другой стороны, обычно увязали в попытках строго определить границы между стадиями. Например, типичная проектная команда могла добиваться замораживания

100% основных требований до перехода к проектированию либо могла пытаться создать проектную документацию во всех деталях, прежде чем перейти к кодированию. В результате чрезмерные усилия распылялись по мелочам, в то время как прогресс в принятии важных рабочих решений тормозился или даже останавливался.

В современном процессе стараются не давать стадиям названия в соответствии с доминирующими видами деятельности. Названия стадий — начальная стадия, уточнение, конструирование, ввод в действие — скорее определяют состояние проекта, чем последовательность действий, аналогичную водопадной модели. Делается это с намерением явно признать непрерывность работ во всех стадиях и отойти от последовательного движения от требований к проектированию, кодированию, тестированию и внедрению.

8.1 РАБОЧИЕ ПРОЦЕССЫ

В предыдущих главах были введены понятия «макропроцесс жизненного цикла» и «фундаментальные комплекты рабочих продуктов». Макропроцесс состоит из дискретных стадий и итераций, но не из дискретных видов работ. Непрерывность различных видов работ присуща каждой стадии и итерации. Процессы более низкого уровня описываются как *микропроцессы*, или *рабочие процессы* (workflow), в результате которых возникают рабочие продукты. Термин «рабочий процесс» используется для обозначения потока связных и в основном последовательных действий. Рабочим процессам соответствуют рабочие продукты (см. главу 6) и команды, работающие над проектом (см. главу 11). Существуют семь рабочих процессов самого верхнего уровня:

1. Процесс управления проектом: контроль за ходом работ и гарантия условий достижения успеха для всех заинтересованных сторон.
2. Процесс создания рабочей среды: автоматизация процесса и развитие среды сопровождения и эксплуатации.
3. Процесс управления требованиями: анализ проблемной области и совершенствование рабочих продуктов требований.
4. Процесс проектирования: моделирование решения и совершенствование архитектуры и рабочих продуктов проектирования.
5. Процесс реализации: программирование компонентов и совершенствование рабочих продуктов реализации и внедрения.
6. Процесс оценки: оценки тенденций и качества продукта.
7. Процесс внедрения: передача конечных продуктов пользователю.

На рис. 8.1 показаны относительные уровни трудозатрат, которые, как предполагается, потребуются на протяжении различных стадий для каждого из рабочих процессов верхнего уровня. Здесь дается одно из ключевых представлений современной схемы процесса и предлагается точка зрения, относительно которой мы будем обсуждать некоторые из базовых принципов, введенных в главе 4.

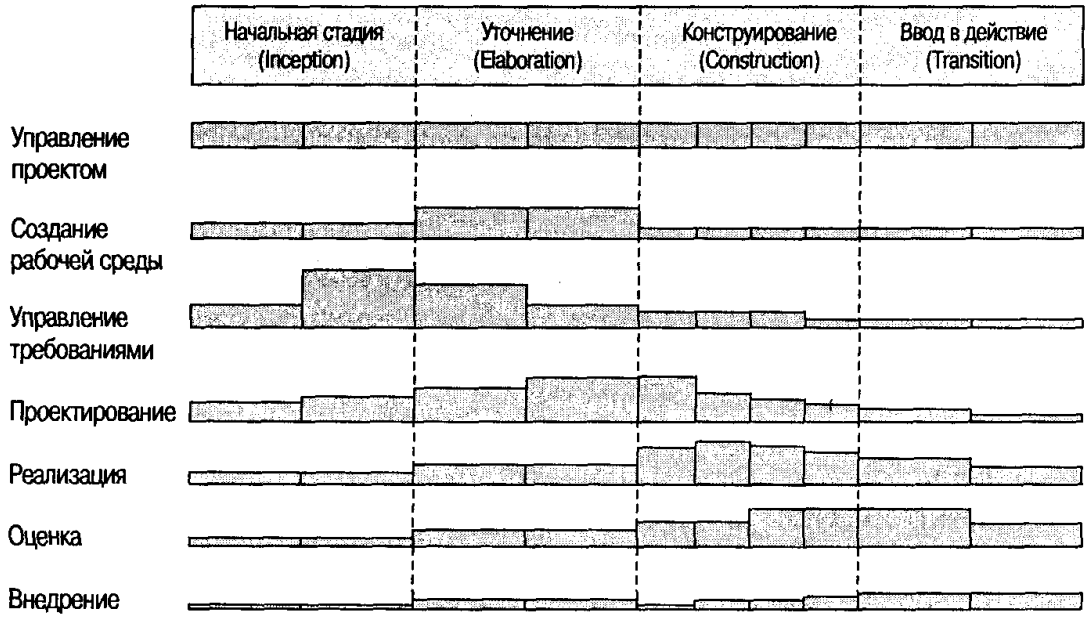


Рис. 8.1. *Интенсивность отдельных видов работ на протяжении различных стадий жизненного цикла*

1. ***Подход с упреждающей разработкой архитектуры.*** Всесторонний анализ требований, проектирование, реализация и оценка выполняются до начала стадии конструирования, когда в центре внимания находится полномасштабная реализация. Внимание, уделяемое на ранних стадиях жизненного цикла реализации и тестированию архитектуры, должно предшествовать как началу полномасштабной разработки и тестированию всех компонентов, так и рассмотрению вопросов завершенности и качества функциональных возможностей продукта.

2. ***Итерационность процесса жизненного цикла.*** На рис. 8.1 для каждой стадии описывается как минимум по две итерации для каждого рабочего процесса. Предполагается, что это значение является декларативным, а не предписывающим. Для некоторых проектов может потребоваться всего одна итерация на протяжении стадии, для других — несколько итераций. Суть заключается в том, что для отдельных работ и рабочих продуктов любого рабочего процесса может понадобиться более одного прохода для достижения адекватных результатов.

3. ***«Круговая» разработка.*** Появление рабочей среды на верхнем уровне процесса создания весьма важно. Среда представляет собой осязаемое воплощение процесса, методов и нотаций, используемых для создания рабочих продуктов проекта.

4. ***Подход, основанный на демонстрациях.*** Работы по реализации и оценке инициируются на ранних стадиях жизненного цикла, отражая то внимание, которое уделяется созданию исполняемых компонентов архитектуры.

Некоторые ключевые для традиционного процесса темы не попали в перечень основных рабочих процессов. Это отсутствие тоже важно. Опущена документация – потому, что она должна являться побочным продуктом других видов работ. Контроль качества исключен по той причине, что он входит во все виды работ, а не выделяется в отдельный рабочий процесс, который выполняется независимо от разработки или управления проектом.

В таблице 8.1 показано распределение рабочих продуктов и основного внимания, уделяемого каждому рабочему процессу на всех стадиях жизненного цикла: начальной стадии, уточнения, конструирования и ввода в действие.

Таблица 8.1.
Рабочие продукты и то, чему уделяется основное внимание на протяжении жизненного цикла для каждого рабочего процесса

Рабочий процесс	Рабочие продукты	Основное внимание на стадиях жизненного цикла
Управление проектом	Бизнес-план План разработки ПО Оценки состояния Концепция Декомпозиция работ	Начальная стадия: подготовка бизнес-плана и концепции Уточнение: разработка плана Конструирование: мониторинг и контроль разработки Ввод в действие: мониторинг и контроль внедрения
Создание рабочей среды	Среда База данных с запросами на внесение изменений	Начальная стадия: определение среды разработки и инфраструктуры управления изменениями Уточнение: инсталляция среды разработки и формирование базы данных для управления изменениями Конструирование: сопровождение среды разработки и базы данных управления изменениями Ввод в действие: Сопровождение среды внедрения и базы данных управления изменениями
Управление требованиями	Комплект требований Спецификации версии Концепция	Начальная стадия: определение аспектов функционирования системы Уточнение: определение целей архитектуры Конструирование: определение целей итерации Ввод в действие: уточнение целей версии
Проектирование	Комплект проектирования Описание архитектуры	Начальная стадия: формулирование концепции архитектуры Уточнение: формирование базовой архитектуры Конструирование: проектирование компонентов Ввод в действие: уточнение архитектуры и компонентов

Таблица 8.1. (продолжение)
Рабочие продукты и то, чему уделяется основное внимание на протяжении жизненного цикла для каждого рабочего процесса

Рабочий процесс	Рабочие продукты	Основное внимание на стадиях жизненного цикла
Реализация	Комплект реализации Комплект внедрения	Начальная стадия: поддержка прототипов архитектуры Уточнение: создание базовой архитектуры Конструирование: создание всех необходимых компонентов Ввод в действие: сопровождение компонентов
Оценка	Спецификации версии Описания версии Руководство пользователя Комплект внедрения	Начальная стадия: планы оценки, концепция, прототипы Уточнение: архитектура оценок Конструирование: оценка промежуточных версий Ввод в действие: оценка версий продукта
Внедрение	Комплект внедрения	Начальная стадия: анализ сообщества пользователей Уточнение: определение руководства пользователя Конструирование: подготовка материалов для ввода в действие Ввод в действие: передача продукта пользователю

Рабочие процессы управления требованиями, проектирования, реализации и оценки подробно освещаются в других книгах по UML. Основное внимание процессам управления проектом и создания рабочей среды уделено в части III. Обсуждение внедрения сведено к минимуму, поскольку обычно оно весьма специфично для каждого проекта.

8.2 РАБОЧИЕ ПРОЦЕССЫ В РАМКАХ ОТДЕЛЬНОЙ ИТЕРАЦИИ

Итерация состоит из множества присутствующих в различных пропорциях работ, последовательность выполнения которых жестко не задана. Эти пропорции зависят от места итерации в цикле разработки. Каждая итерация определяется в терминах набора сценариев использования. Компоненты, необходимые для реализации всех выбранных сценариев, разрабатываются и интегрируются с результатами предыдущих итераций. Рабочий процесс в рамках отдельной итерации (см. рис. 8.2) обычно включает в себя следующую последовательность:

- Управление проектом: планирование итерации с целью определения содержимого версии и разработки подробного плана итерации; распределение пакетов заданий или задач между командами.

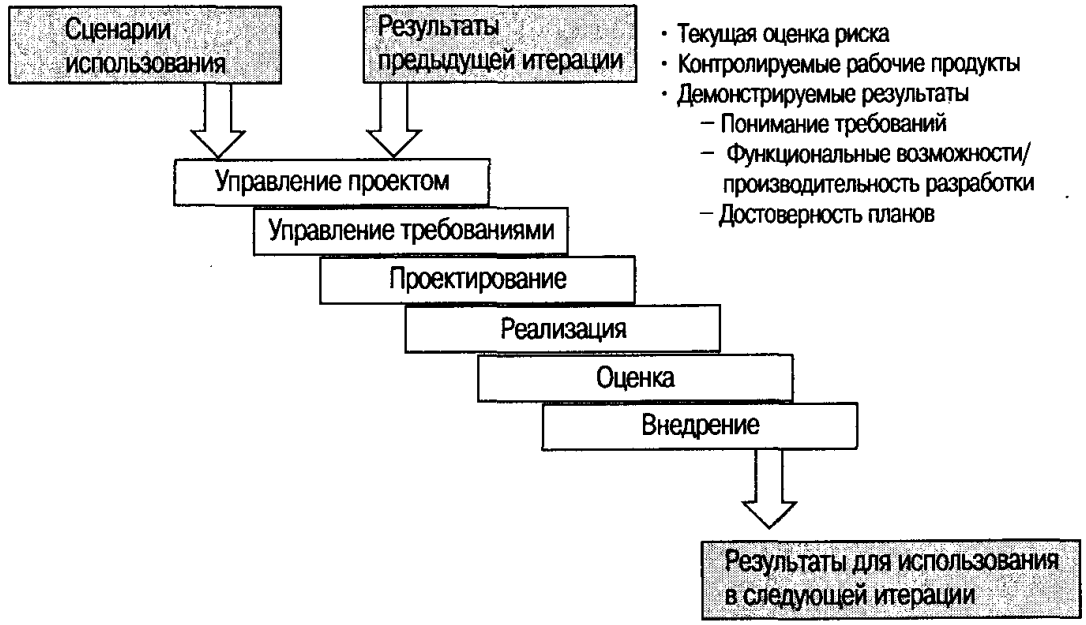


Рис. 8.2. *Рабочие процессы в рамках одной итерации*

- Создание рабочей среды: приведение базы данных с запросами на внесение изменений в ПО в соответствие с изменениями всех компонентов продукта, тестов и среды.
- Управление требованиями: анализ комплектов рабочих продуктов по базовым планам, архитектуре и требованиям для полной проработки вариантов использования, предназначенных для демонстрации в конце данной итерации, и их критериев оценки; обновление всех рабочих продуктов комплекта требований с тем, чтобы они отражали изменения, внесенные в результате данной итерации.
- Проектирование: внесение изменений в комплекты рабочих продуктов по базовой архитектуре и проектным решениям для приведения в полное соответствие компонентов проектной и тестовой модели, необходимых для демонстрации соответствия критериям данной итерации; обновление всех рабочих продуктов комплекта проектирования, чтобы они отражали все изменения, внесенные в результате данной итерации.
- Реализация: разработка или получение любых новых компонентов, а также расширение и модификация любых уже существующих компонентов с целью демонстрации соответствия критериям данной итерации; интеграция и тестирование всех новых и модифицированных компонентов вместе с предыдущими версиями.
- Оценка: оценка результатов итерации, включая проверку на соответствие критериям оценки и качеству существующей основы; выявление всех необходимых доработок и принятие решения, следует ли выполнять их перед внедрением данной версии или можно перенести на следующую версию; оценка результатов для улучшения плана последующих итераций.

- Внедрение: передача версии либо сторонней организации (пользователю, независимому подрядчику для выполнения верификации и аттестации или регулирующему органу), либо на проведение «вскрытия» внутри организации с тем, чтобы можно было извлечь уроки и учесть их в следующей итерации.

При любой последовательности работ по созданию ПО многие виды деятельности выполняются параллельно. Например, анализ требований не является одним непрерывным аккордом; он пересекается с управлением проектом, проектированием, реализацией и т.д.

В итерациях на начальной стадии и стадии уточнения основное внимание уделяется управлению проектом, требованиям и проектированию. В итерациях на стадии конструирования основное внимание уделяется проектированию, реализации и оценке. В итерациях на стадии ввода в действие основное внимание уделяется оценке и внедрению. То, чему уделяется внимание на протяжении всего жизненного цикла, показано на рис. 8.3.

Рис. 8.3. *На что обращается основное внимание в итерациях в течение всего жизненного цикла*

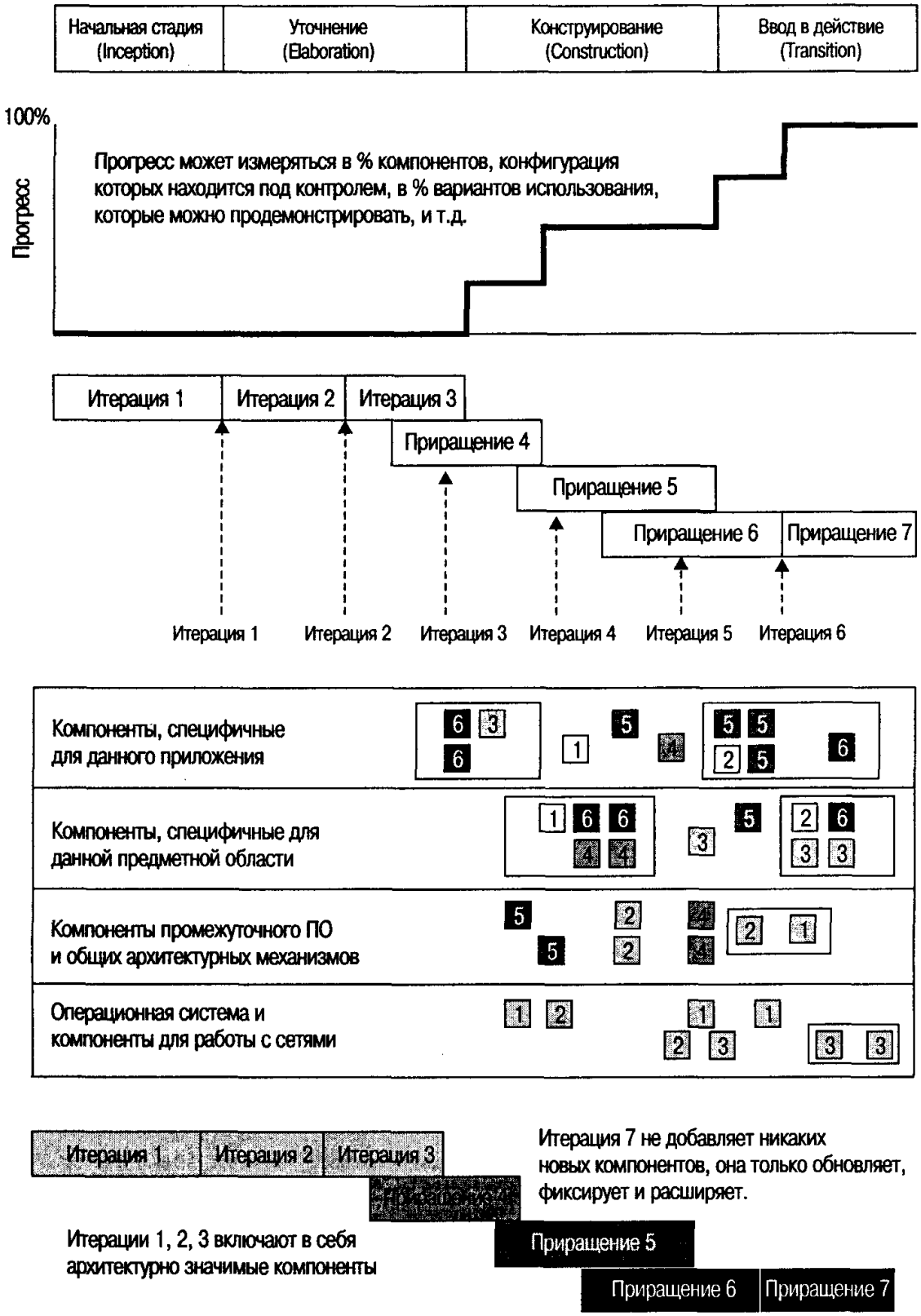


Рис. 8.4. *Типичная последовательность создания системы, присущая послойной архитектуре*

Эти описания весьма упрощенны. На практике различная последовательность итераций и их взаимное наложение гораздо сложнее. Термины «итерация» и «приращение» имеют дело с некими прагматическими подходами. Итерация обозначает состояние архитектуры в целом и завершенную систему, готовую к внедрению. Приращение обозначает текущую работу в развитии, которая в сочетании с предшествующей итерацией образует последующую итерацию. Представленный на рис. 8.4 пример простого жизненного цикла разработки показывает различия между итерацией и приращением. Этот пример иллюстрирует также типичную последовательность создания с точки зрения абстрактной послойной архитектуры.

Глава 9

Контрольные точки процесса

Ключевые моменты

▲ Для согласования ожиданий заинтересованных сторон на протяжении всего жизненного цикла используются три последовательности контрольных точек проекта: основные точки, второстепенные точки и оценки состояния.

▲ Наиболее важной основной точкой обычно является момент перехода проекта со стадии уточнения в стадию конструирования.

▲ Формат и содержание второстепенных точек в сильной степени зависит от конкретного проекта и от культуры организации.

▲ Периодическая оценка состояния важна с точки зрения постоянного внимания, которое уделяется ходу проекта и его приоритетам.

Представляется важным наличие наглядных контрольных точек на протяжении жизненного цикла, когда различные заинтересованные стороны могли бы встречаться лицом к лицу и обсуждать достигнутый прогресс и планы. Целью таких мероприятий является не только демонстрация хода выполнения проекта, но и решение следующих задач:

- Согласование ожиданий заинтересованных сторон и трех составляющих проекта: требований, проектных решений и планов.
- Приведение взаимосвязанных рабочих продуктов в непротиворечивое и сбалансированное состояние.
- Выявление важных рисков, проблем и невыполнимых условий.
- Выполнение глобальной оценки всего жизненного цикла, а не только какого-либо промежуточного продукта или текущей ситуации в какой-то одной перспективе.

Необходимо четко определить, какие существуют ожидания и каковы осязаемые результаты для каждой контрольной точки. Это не исключает новых договоренностей относительно целей, достигаемых в каждой контрольной точке, по мере того, как в рамках проекта понимание согласованности между требованиями, проектными решениями и планом выходит на новый уровень.

В процессе руководства проектом проводятся совместные оценки трех разных типов:

1. *Основные контрольные точки.* Эти мероприятия на уровне всей системы проводятся в конце каждой стадии разработки. Они позволяют выявить крупные проблемы, согласовать точки зрения управления и разработки и подтвердить, что цели данной стадии достигнуты.
2. *Второстепенные контрольные точки.* Эти мероприятия, в фокусе внимания которых находятся итерации, проводятся для детальной проверки содержания итерации и для санкционирования дальнейшей работы.
3. *Оценки состояния.* Эти периодические мероприятия позволяют руководству регулярно вникать в суть достигнутых успехов.

Каждая из четырех стадий — начальная, уточнение, конструирование и ввод в действие — состоит из одной или более итераций и завершается основной контрольной точкой, когда планируемая техническая характеристика оказывается воплощенной в демонстрируемой форме. Итерация представляет собой циклический вид деятельности, для которого четко определен промежуточный результат — второстепенная контрольная точка,— связанный с двумя видами рабочих продуктов: спецификация версии (критерии оценки и план) и описание версии (результаты). Для основных контрольных точек в конце каждой стадии используются формальные, одобренные заинтересованными сторонами критерии оценки и описания версий; для второстепенных контрольных точек применяются неформальные — на усмотрение команды разработчиков — редакции рабочий продуктов.

Уровень мероприятия и количество контрольных точек меняется в зависимости от таких параметров, как масштаб, количество заинтересованных сторон, состояние бизнеса, технический риск и чувствительность проекта к изменениям затрат и сроков. Для большинства проектов нужно установить все четыре основные контрольные точки. Только в исключительных случаях следует добавлять другие основные контрольные точки или оперировать меньшим их числом. (Для проекта государственной важности, привлекающего широкое внимание, их число можно увеличить; в случае научного эксперимента для внутреннего использования их может быть меньше.) В более простых проектах для контроля за промежуточными результатами может понадобиться меньшее число второстепенных контрольных точек или они не потребуются вовсе, а периодичность оценок состояния может быть небольшой (например, ежеквартальной). На рис. 9.1 показана типичная последовательность контрольных точек для сравнительно большого проекта.

Описания этого раздела напоминают подход с «якорными» точками жизненного цикла, рассмотренный в статье «Anchoring the Software Process» [Boehm, 1996]. Четыре основные контрольные точки располагаются в точках перехода между стадиями жизненного цикла. Они могут использоваться в самых разнообразных моделях процесса, включая традиционную водопадную модель. В итерационной модели основные

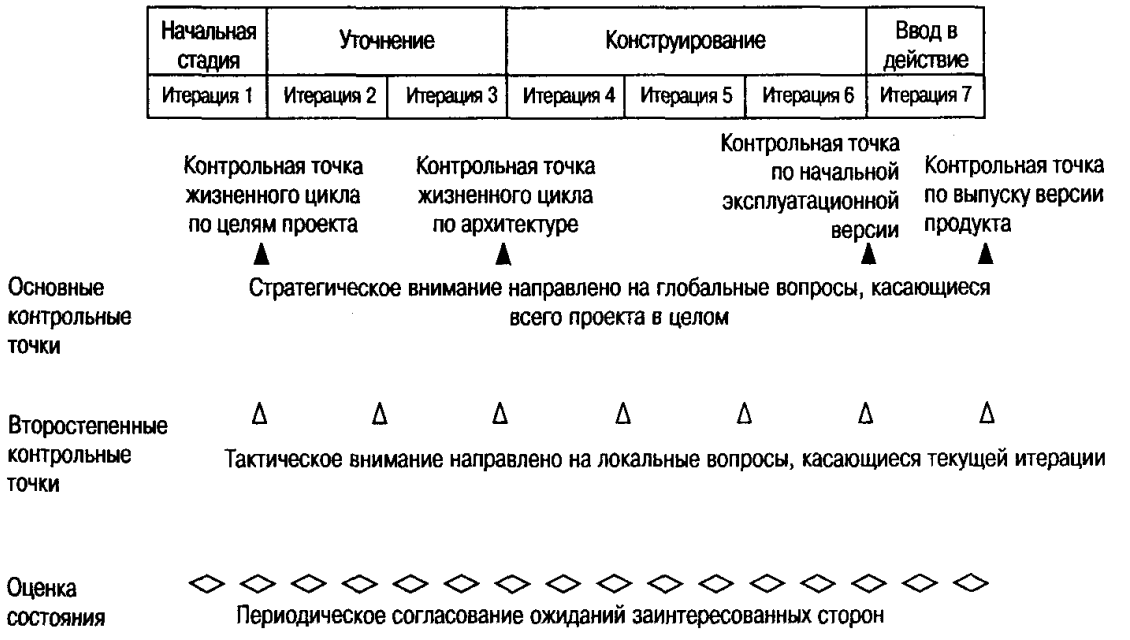


Рис. 9.1. *Типичная последовательность контрольных точек на протяжении всего жизненного цикла*

контрольные точки применяются для достижения согласованности среди всех заинтересованных сторон относительно текущего состояния проекта. Понятия разных заинтересованных сторон могут сильно различаться:

- Заказчики: оценки сроков и бюджета, осуществимость, оценка рисков, понимание требований, прогресс, совместимость продуктов одной линии
- Пользователи: непротиворечивость требований и сценариев использования, потенциальная адаптируемость, характеристики качества
- Архитекторы и системотехники: совместимость продуктов одной линии, изменение требований, анализ согласований, завершенность и непротиворечивость, баланс между риском, качеством и применимостью
- Разработчики: детальность требований и описаний сценариев использования, основа для выбора или разработки компонентов, разрешение рисков при разработке, достаточность среды разработки
- Служба сопровождения: достаточность продукта и сопровождающей его документации, понятность, возможность совместного использования с уже существующими системами, достаточность среды сопровождения
- Другие: возможно множество других точек зрения таких заинтересованных сторон, как регулирующие органы, независимые подрядчики для выполнения верификации и аттестации, инвесторы, вложившие капитал в предприятие, субподрядчики, смежники и команды, осуществляющие продажу и маркетинг

Контрольные точки, описываемые в этом разделе, могут проводиться как единая постоянно действующая конференция всех заинтересованных сторон либо в виде последовательных рассмотрений различных рабочих продуктов преимущественно в онлайновом режиме. Имеются существенные различия в уровне формальности этих мероприятий, что зависит от некоторых факторов, обсуждаемых в главе 14. Смыслом каждой основной контрольной точки является подтверждение непротиворечивости различных рабочих продуктов, а также получение гарантий того, что понимание требований, планы, касающиеся всего жизненного цикла, форма, функции и качество продукта развиваются на сбалансированных уровнях детализации. Распределение информации между основными контрольными точками представлено в таблице 9.1.

Таблица 9.1.
Общее состояние планов, требований и продуктов при достижении основных контрольных точек

Контрольные точки	Планы	Понимание проблемной области (требования)	Прогресс в области решений (программный продукт)
Контрольная точка жизненного цикла по целям	Определение ответственности заинтересованных сторон Приблизительный план жизненного цикла Точный план стадии уточнения	Общая концепция, включая направления развития, показатели качества и приоритеты Модель вариантов использования	Демонстрация по крайней мере одной допустимой архитектуры Решения о создании/покупке/повторном использовании Начальная проектная модель
Контрольная точка жизненного цикла по архитектуре	Точный план стадии конструирования (список рабочих продуктов, распределение работ) Приблизительный план стадии ввода в действие	Стабильная концепция и модель вариантов использования Критерии оценки для версий стадии конструирования, для начальной эксплуатационной версии Черновой вариант руководства пользователя	Стабильный комплект проектирования Решения о создании/покупке/повторном использовании Критичные прототипы компонентов
Контрольная точка по начальной эксплуатационной версии	Точный план стадии ввода в действие	Критерии приемки версии продукта Руководство пользователя в виде, пригодном для выпуска	Стабильный комплект реализации Наиболее важные свойства и функциональные возможности Объективное подробное рассмотрение качеств продукта

Таблица 9.1. (продолжение)
Общее состояние планов, требований и продуктов при достижении основных контрольных точек

Контрольные точки	Планы	Понимание проблемной области (требования)	Прогресс в области решений (программный продукт)
Контрольная точка по выпуску версии продукта	План создания продукта следующего поколения	Окончательный вариант руководства пользователя	Стабильный комплект внедрения Полный набор возможностей Соответствующее качество

Контрольная точка жизненного цикла по целям

Контрольная точка жизненного цикла по целям находится в конце начальной стадии. Ее задачей является представить всем заинтересованным сторонам рекомендации относительно того, как приступать к разработке. Рассматриваются план, оценочные стоимость и сроки, а также ожидаемые прибыли и экономия средств. Обсуждаются общая концепция и критичные проблемы, имеющие отношение к требованиям и аспектам функционирования будущей системы. Черновой вариант документа по архитектуре и демонстрация прототипа архитектуры служат свидетельством завершенности общей концепции и плана создания ПО. Успешно достигнутая контрольная точка по целям жизненного цикла приводит к тому, что переход к стадии уточнения будет санкционирован всеми заинтересованными сторонами.

Контрольная точка жизненного цикла по архитектуре

Контрольная точка жизненного цикла по архитектуре находится в конце стадии уточнения. Ее основная цель — продемонстрировать всем заинтересованным сторонам архитектуру в работе. Представляется на утверждение более детальный план стадии конструирования. Обсуждаются критичные проблемы, касающиеся требований и аспектов функционирования. Это рассмотрение также приводит к консенсусу по базовой архитектуре, основам общей концепции, основам плана разработки ПО и критериям оценки для контрольной точки по начальной эксплуатационной версии. Базовая архитектура состоит из представления, пригодного для восприятия человеком (документ), и комплекта программных компонентов с управляемой конфигурацией, содержащего рабочие продукты проектирования. Успешно достигнутая контрольная точка жизненного цикла по архитектуре приводит к тому, что заинтересованные стороны санкционируют переход к стадии конструирования.

Поскольку наиболее важной основной контрольной точкой обычно является событие, которое переводит проект из стадии уточнения в стадию конструирования, общее содержание типичной контрольной точки

проработано здесь более подробно. С позиций руководства и сторон, заключивших контракт, эта основная контрольная точка соответствует достижению такого состояния в разработке ПО, при котором исследования и разработка завершаются и начинается стадия изготовления. Проект, готовый к такому переходу, обладает следующими характеристиками:

- Критически важные варианты использования определены, согласованы между всеми заинтересованными сторонами и запрограммированы в виде набора сценариев для оценки архитектуры.
- Определены основы стабильной архитектуры (под управлением конфигурацией) в формате исходного языка. Стабильность в данном случае означает, что важнейшие качества архитектуры (производительность, устойчивость, масштабируемость, адаптируемость) продемонстрированы по отношению к критически важным вариантам использования в достаточной мере, чтобы удовлетворять всем основным требованиям, а также рискам разработки и планирования. (Риски могут еще иметь место, но путь их разрешения должен быть определен.)
- Структура рисков вполне понятна. Хотя полного разрешения всех рисков и не требуется, должно быть достигнуто понимание всеми заинтересованными сторонами наиболее значительных рисков, которые могут иметь серьезные последствия, при этом должны быть полностью проработаны планы их снижения.
- План разработки для стадий конструирования и ввода в действие определяется довольно точно, так, чтобы итерации конструирования могли производиться с предсказуемыми результатами. В данном случае предсказуемость означает, что организация, ведущая разработку, выполняет некоторую часть работы с фиксированной стоимостью, которая может быть передана пользователю не позднее, чем через один год.

Содержание этой контрольной точки может меняться в зависимости от предметной области проекта. Как минимум в нее будет входить следующее:

- Представление и обзор текущего состояния проекта
- Совокупность информации, касающейся разработки, находящейся под управлением конфигурацией и доступной в электронном виде или в виде твердой копии
- Демонстрация возможностей

Технические данные, приведенные на рис. 9.2, следует рассматривать к моменту достижения контрольной точки жизненного цикла по архитектуре. На рис. 9.3 представлена стандартная программа для этой контрольной точки.

I. Требования

A. Модель вариантов использования
B. Документ с общей концепцией (текст, варианты использования)
C. Критерии оценки для стадии уточнения (текст, сценарии)

II. Архитектура

A. Проектное представление (объектные модели)
B. Представление процессов (при необходимости — разбивка на модули, структура исполняемого кода)
C. Представление компонентов (структура подсистем, идентификация создаваемых/покупаемых/повторно используемых компонентов)
D. Представление внедрения (физическое размещение компонентов, структура исполняемого кода)
E. Представление вариантов использования (структура тестовых вариантов, ожидаемые результаты тестов)
1. Черновой вариант руководства пользователя

III. Библиотеки исходных текстов и исполняемых компонентов

A. Компоненты продукта
B. Компоненты для тестирования
C. Компоненты среды и инструментария

Рис. 9.2. *Рабочие продукты разработки, доступные при достижении контрольной точки жизненного цикла по архитектуре*

Контрольная точка по начальной эксплуатационной версии

Контрольная точка по начальной эксплуатационной версии находится в конце стадии конструирования. Ее цель – оценка готовности ПО к передаче его в эксплуатацию заказчику/пользователю и санкционирование начала приемочного тестирования. Обсуждаются проблемы, касающиеся инструкций по инсталляции, описаний версий ПО, руководств пользователя и оператора, а также возможности организации-разработчика осуществлять сопровождение продукта у пользователя. Рассматриваются характеристики качества ПО для определения его достаточности. Оценивается готовность тестовой среды и тестового ПО к приемочному тестированию. Приемочное тестирование может проводиться по нарастающей на протяжении многих итераций либо может быть выполнено полностью в течение стадии внедрения. Начало стадии внедрения необязательно должно являться завершением стадии конструирования. Эти стадии обычно перекрывают друг друга до тех пор, пока первоначальный продукт не будет передан пользователю для самостоятельной работы.

Контрольная точка по выпуску версии продукта

Контрольная точка по выпуску версии продукта находится в конце стадии внедрения. Ее целью являются оценка завершенности ПО и его передача организации, ответственной за сопровождение, если таковая

Программа презентации

I. Область действия и цели

A. Демонстрационный обзор

II. Оценка требований

A. Общая концепция проекта и варианты использования

B. Главные сценарии и критерии оценки

III. Оценка архитектуры

A. Прогресс

1. Параметры базовой архитектуры (прогресс на сегодняшний день и основы для оценки стабильности архитектуры в будущем, отбраковка и доработка)
2. Базовая оценка параметров разработки (для выполнения оценок прогресса в будущем)
3. Базовая оценка параметров тестирования (для выполнения оценок прогресса команды тестировщиков в будущем)

B. Качество

1. Характеристики архитектуры (итоги демонстрации возможностей относительно критериев оценки)
2. Производительность (итоги демонстрации возможностей относительно критериев оценки)
3. Выявленные риски в архитектуре и планы их разрешения
4. Бюджетная оценка и соглашения относительно создания/покупки/повторного использования

IV. Оценка плана стадии конструирования

A. Содержание итерации и распределение вариантов использования

B. Подробный план последующих итераций и критерии оценки

C. Сроки/затраты на стадии уточнения

D. План требуемых ресурсов для стадии конструирования и основы оценок

E. Оценка рисков

Программа демонстрации

I. Критерии оценки

II. Подведение итогов по архитектуре

III. Подведение итогов по среде демонстрации

IV. Запрограммированные сценарии демонстрации

V. Результаты оценок и последующие вопросы

Рис. 9.3. *Стандартные программы для контрольной точки жизненного цикла по архитектуре*

имеется. Рассматриваются результаты приемочного тестирования, а также обсуждаются все вопросы, оставшиеся открытыми. Среди них могут быть вопросы по инструкциям инсталляции, описаниям версий ПО, руководствам пользователя и оператора, руководствам по сопровождению ПО и по инсталляции среды разработки там, где будет выполняться сопровождение. Рассматриваются характеристики качества ПО на предмет его достаточности для передачи организации, ответственной за сопровождение.

9.2 ВТОРОСТЕПЕННЫЕ КОНТРОЛЬНЫЕ ТОЧКИ

Количество необходимых для данной итерации неформальных контрольных точек зависит от содержания и длины итерации. Для большинства итераций продолжительностью от одного до шести месяцев требуются только две второстепенные контрольные точки: рассмотрение готовности к итерации и рассмотрение оценки итерации. Для более длительных итераций может потребоваться больше промежуточных точек. Например, для проектов с формальными процедурами тестирования, которые должны проводиться в присутствии других заинтересованных сторон, рассмотрение готовности к тестированию может проводиться тогда же, когда рассматриваются и принимаются планы тестирования. В крупномасштабных, не имеющих прецедентов проектах также могут использоваться промежуточные оценки хода разработки в качестве стимуляторов для оценок прогресса и распространения по всему проекту.

Итерации не одинаковы. Итерация может иметь совершенно различные формы и приоритеты, зависящие от того, на какой стадии жизненного цикла находится проект. В ранних итерациях основное внимание уделяется анализу и проектированию со значительными элементами новаторства, экспериментирования и оценки рисков. Более поздние итерации в гораздо большей степени сосредоточивают свое внимание на завершенности, непротиворечивости, применимости и управлении изменениями. Контрольные точки внутри итераций и связанные с ними критерии оценки фокусируют различные виды деятельности на те приоритеты проекта, которые описаны в общем плане создания ПО, бизнес-плане и общей концепции.

- **Рассмотрение готовности к итерации.** Эта неформальная контрольная точка находится в начале каждой итерации и служит для рассмотрения подробного плана итерации и критериев оценки, которые были отнесены к данной итерации.
- **Рассмотрение оценки итерации.** Эта неформальная контрольная точка находится в конце каждой итерации и служит для оценки той степени, в которой итерация достигла поставленных целей и удовлетворяет своим критериям оценки, для рассмотрения результатов итерации, для рассмотрения результатов квалификационного тестирования (если тестирование является частью итерации), для определения объема доработок, которые придется выполнять, и для рассмотрения влияния результатов итерации на план последующих итераций.

Формат и содержание этих второстепенных контрольных точек сильно зависят от особенностей проекта и организационной культуры. Рис. 9.4 определяет различные второстепенные контрольные точки, которые следует принять во внимание при планировании проекта.

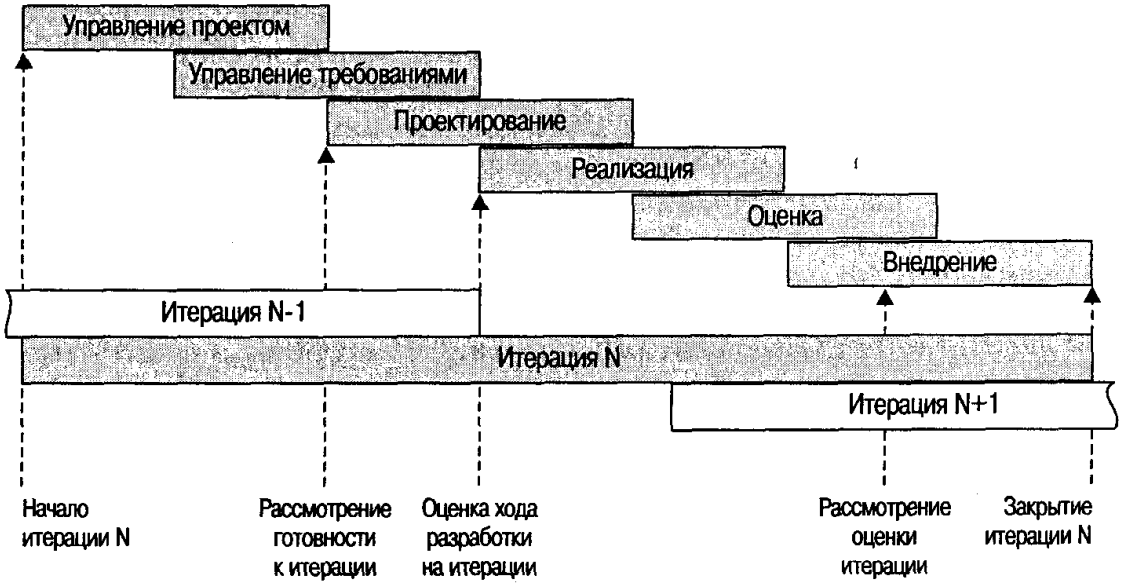


Рис. 9.4. *Типичные второстепенные контрольные точки жизненного цикла итерации*

9.3 ПЕРИОДИЧЕСКИЕ ОЦЕНКИ СОСТОЯНИЯ

Управление рисками требует постоянного внимания ко всем оказывающим влияние друг на друга видам деятельности, выполняемым при разработке. Периодические оценки состояния — это рассмотрения на уровне руководства, проводимые через регулярные промежутки времени (ежемесячно, ежеквартально) для определения прогресса и качественных показателей, для подтверждения постоянного внимания к динамике проекта и для поддержания открытого взаимодействия между всеми заинтересованными сторонами. Главной целью этих оценок является подтверждение того, что ожидания всех заинтересованных сторон (подрядчик, заказчик, пользователь, субподрядчик) согласованы и непротиворечивы.

Периодические оценки состояния служат в качестве «моментальных снимков» состояния проекта. Хотя их период может варьироваться, повторяющиеся события заставляют фиксировать и документировать историю проекта. Оценка состояния предполагает следующее:

- Наличие механизма для открытого обсуждения, обмена мнениями и разрешения проблем управления, технических проблем и рисков проекта
- Получение объективных данных непосредственно из текущей деятельности и изменяющихся конфигураций продукта

- Наличие механизма распространения информации о процессе, прогрессе, качественных тенденциях, практике и опыте между всеми заинтересованными сторонами в форме свободной дискуссии

Повторяющиеся среди неудачных проектов проблемы включают в себя оценки состояния, которые (1) требовали выполнения действий, оказывавшихся весьма накладными, поскольку работа, связанная с оценкой состояния, отделялась от повседневной работы, и (2) часто отменялись по причине более приоритетных проблем, требовавших решения. В удачных проектах выполнялись оценки состояния, которые (1) требовали небольших накладных расходов, поскольку необходимые рабочие продукты уже существовали в виде текущих данных, и (2) редко отменялись, потому что считались чрезвычайно важными.

Периодические оценки состояния важны с точки зрения постоянного внимания, уделяемого ходу проекта и его приоритетам. Они заставляют менеджера проекта периодически собирать и анализировать данные, форсируют внимательное рассмотрение со стороны и способствуют распространению лучшей практики среди остальных заинтересованных сторон. Стандартизируя формат рассмотрения и изучаемые характеристики, организация получает также возможность выполнять сравнение различных проектов между собой и распространять самую удачную практику более эффективно.

Стандартное содержание периодических оценок состояния должно включать в себя разделы, обозначенные в таблице 9.2. Единственным, что менеджеру проекта придется извлекать из отбраковок, является оценка десяти самых высоких рисков. Но даже это будет преимущественно обновление предыдущей оценки. Существует хорошее эмпирическое правило, которое гласит, что диаграммы оценки состояния должны с легкостью создаваться менеджером проекта за один день. Такие минимальные усилия оказываются возможными, если данные хранятся в электронном виде. Тема «Технический прогресс», приведенная в таблице, обсуждается в главе 13.

Таблица 9.2.
Стандартное содержание оценки состояния

Тема	Содержание
Персонал	План по найму в сравнении с реальностью Нестыковки, дополнения
Финансовые тенденции	План затрат по сравнению с реальными затратами для предыдущей, текущей и будущей основных контрольных точек Прогноз прибыли
10 самых опасных рисков	Проблемы и планы разрешения рисков Количественные показатели (затраты, время, качество), связанные с возможными дефектами
Технический прогресс	Планирование базовой конфигурации для основных контрольных точек Состояние и показатели управления созданием ПО Текущие тенденции изменений#Оценки тестирования и качества

Таблица 9.2. (продолжение)
Стандартное содержание оценки состояния

Тема	Содержание
Планы и результаты основных контрольных точек	План, сроки и риски очередной основной контрольной точки Результаты типа «прошел/не прошел» для всех критериев приемки
Область действия продукта в целом	Общие изменения размера, процесса роста и критериев приемки

Часть III

Дисциплины управления процессом создания ПО

Глава 10 ◆ **Планирование итерационного процесса**

Глава 11 ◆ **Проектные организации и распределение обязанностей**

Глава 12 ◆ **Автоматизация процесса**

Глава 13 ◆ **Контроль за проектом и инструментальное оснащение процесса**

Глава 14 ◆ **Адаптация процесса**

Работы по управлению созданием ПО охватывают широкий спектр областей. В главах части III обсуждаются основные дисциплины, необходимые для эффективного управления: планирование, организация, автоматизация и контроль над проектом. Все эти дисциплины сложно описать в общих словах. Следовательно, появляется еще одна важная дисциплина — адаптация процесса к специфическим требованиям данного проекта.

Планирование является главной проблемой управления проектом. Основная задача заключается в разработке плана, позволяющего распределить имеющиеся в наличии ресурсы таким образом, чтобы создать наиболее благоприятные условия для всех заинтересованных сторон. Дисциплина «организация проекта» касается управления людьми, т.е. формирования команд и распределения ответственности. Автоматизация процесса разработки посредством хранения рабочих продуктов в электронном хранилище (репозитории) создает основу для инструментального оснащения процесса. Различные виды деятельности по контролю над проектом являются его «органами чувств». Они применяются для оценки состояния плана, качества рабочих продуктов и необходимости внесения изменений в рабочие продукты комплекта управления, которые определяют ожидания всех заинтересованных сторон.

Глава 10

Планирование итерационного процесса

Ключевые моменты

- Проекты могут испытывать недостаток планов и их избыток. Повторим еще раз: глубина планирования и распределение планов между заинтересованными сторонами имеют первостепенную важность.
- Декомпозиция работ является «архитектурой» плана проекта. Необходимо иметь возможность изменять ее, и она должна претерпевать изменения с соответствующей степенью детализации на протяжении всего жизненного цикла.
- Для получения предсказуемых результатов оценка затрат и сроков должна проводиться с использованием методов макроанализа (сверху вниз на уровне проекта) и микроанализа (снизу вверх на уровне отдельных задач).

Так же как и создание ПО, планирование проекта требует итерационного процесса. Так же как и ПО, план является неосязаемым продуктом интеллектуальной деятельности, к которому должны применяться те же самые понятия. Планы проходят стадию разработки, в течение которой они создаются, и производственную стадию, в течение которой они выполняются. По мере того как происходят изменения в понимании проблемной области и области решений, изменения должны вноситься и в планы. Ошибки, допущенные при планировании, аналогичны ошибкам в продукте: чем на более ранних стадиях жизненного цикла они будут разрешены, тем меньшее влияние окажут на успешное осуществление проекта.

Всесторонние планы проекта зависят от множества параметров, каждый из которых может оказывать значительное влияние на направление развития проекта. Тем не менее любому менеджеру проекта дорог хороший совет по планированию, который можно рассматривать в качестве основы для начального составления плана. Эта глава не является ни планом, ни «поваренной книгой» планов, ни рецептом для плана. Она предлагает лишь грубую модель в нескольких измерениях и, возможно, отправную точку для создания плана.

10.1 ДЕКОМПОЗИЦИЯ РАБОТ

Хорошая декомпозиция работ (WBS) и ее синхронизация со схемой процесса – важные факторы успеха проекта. Понятие и практика использования WBS являются вполне устоявшимися, но в публикациях стараются избегать этой темы, и прежде всего потому, что декомпозиция работ зависит от стиля управления проектом, организационной культуры, предпочтений заказчика, финансовых ограничений и от некоторых других трудно поддающихся определению параметров, специфичных для каждого проекта. В книге «Software Engineering Economics» [Boehm, 1981] содержатся основные сведения о декомпозиции работ, ориентированной на ПО.

WBS – это иерархия элементов, которая позволяет разбивать план проекта на отдельные рабочие задания. WBS имеет следующую информационную структуру:

- Общее описание всех значительных работ
- Четкое разбиение на задачи для распределения ответственности
- Схема для определения сроков, финансирования и контроля над расходами

На разбиение всей работы на отдельные задачи может оказывать влияние множество факторов: подсистемы продукта, компоненты, функции, организационные единицы, стадии жизненного цикла, даже география. Большинству систем присуще разбиение первого уровня – на подсистемы. Подсистемы, в свою очередь, разбиваются на компоненты, одним из которых обычно является ПО. В следующем разделе основное внимание уделяется программным элементам WBS независимо от того, является ли создание ПО смыслом всего проекта или одним из компонентов более сложной системы.

10.1.1 Проблемы традиционной WBS

Традиционные декомпозиции работ обычно имеют три фундаментальных порока.

1. Они создаются преждевременно при разработке продукта.
2. Они преждевременно детализируются, планируются и финансируются либо слишком подробно, либо недостаточно подробно.
3. Они специфичны для каждого проекта, поэтому их сравнение для разных проектов обычно затруднительно или невозможно.

Традиционные декомпозиции работ создаются преждевременно на основе проектных решений по разработке продукта. На рис. 10.1 показана типичная традиционная WBS, построенная на основе разбиения архитектуры продукта на подсистемы, которые в свою очередь разбиты на компоненты. Как только эта структура находит свое отражение в WBS и переходит к ответственным менеджерам в виде бюджетов, сроков и ожидаемых отчетов, устанавливается такая основа для планирования, менять которую

оказывается сложно и дорого. WBS является архитектурой финансового плана. В архитектуре ПО следует инкапсулировать компоненты, которые, вероятно, будут изменяться, точно так же необходимо поступать и с архитектурой планирования. Жестко привязывать план к структуре продукта имеет смысл только в том случае, если они оба достаточно созрели для этого. В случае если либо план, либо архитектура будет подвергаться дальнейшим изменениям, желательна менее жесткая зависимость.

- Управление
- Требования и проектные решения системы
- Подсистема 1
  - Компонент 11
    - Требования
    - Проектирование
    - Кодирование
    - Тестирование
    - Документация
  - ...(аналогичная структура для других компонентов)
  - Компонент 1N
    - Требования
    - Проектирование
    - Кодирование
    - Тестирование
    - Документация
  - ...(аналогичная структура для других подсистем)
- Подсистема M
  - Компонент M1
    - Требования
    - Проектирование
    - Кодирование
    - Тестирование
    - Документация
  - ...(аналогичная структура для других компонентов)
  - Компонент MN
    - Требования
    - Проектирование
    - Кодирование
    - Тестирование
    - Документация
- Интеграция и тестирование
  - Планирование тестирования
  - Подготовка процедур тестирования
  - Тестирование Отчеты о тестировании
- Другие вопросы сопровождения
  - Контроль конфигурации
  - Проверка качества
  - Системное администрирование

Рис. 10.1. *Традиционная декомпозиция работ, привязанная к структуре продукта*

Традиционные декомпозиции преждевременно детализируются, планируются и финансируются либо слишком подробно, либо недостаточно подробно. В больших проектах существует тенденция к избытку планов, а в малых — к недостатку планов. WBS, приведенная на рис. 10.1, оказывается чрезмерно упрощенной для широкомасштабных проектов, где обычными считаются шесть и более уровней элементов WBS. Команда управления проектом тщательно планирует каждый элемент и определяет основы бюджета и сроки создания ПО для каждого задания на данном уровне детализации. С другой стороны, большинство небольших или «домашних» разработок довольствуется созданием WBS с одним уровнем без сопровождения его какими-либо подробностями. Команда управления планирует и ведет проект, используя приблизительное распределение заданий, нежесткую отчетность по срокам и затратам. Оба этих подхода не являются взвешенными. Вообще говоря, имеет смысл WBS, разработанная, по крайней мере, до двух или трех уровней. В случае крупномасштабных систем может понадобиться несколько дополнительных уровней на более поздних стадиях жизненного цикла. Основной проблемой при включении в план с самого начала слишком большого числа деталей заключается в том, что эти детали не меняются параллельно с уровнем качества плана. Например, не представляется возможным точно расписать на первом месяце — когда составляются основы плана, и до того, как будут разработаны архитектура и сценарии тестирования, — детали работ по тестированию, намеченных к выполнению 18 месяцами позже.

Традиционные декомпозиции работ специфичны для каждого проекта, поэтому сравнение разных проектов обычно оказывается затруднительным или невозможным. В большинстве организаций разрешается в каждом отдельном проекте адаптировать описание присущей ему структуры под стиль менеджера проекта, под запросы заказчиков или под какие-либо другие особенности, специфичные для данного проекта. При отсутствии стандартной структуры WBS чрезвычайно сложно сравнивать планы, финансовые показатели, временные показатели, организационную эффективность, тенденции изменения затрат, производительности или качества для разных проектов. В рамках каждого конкретного проекта работа организуется по-разному, при этом используются различные единицы измерения. На некоторые из приведенных ниже простых вопросов, являющихся критичными для любой программы улучшения организации процесса, большинство работающих над проектами команд, применяющих традиционную декомпозицию работ, не сумеют дать ответа.

- Каково соотношение между производительными видами деятельности (требования, проектирование, реализация, оценка, внедрение) и непроизводительными (управление проектом, создание рабочей среды)?
- Каков процент усилий, затраченных на доработки?
- Каков процент затрат на основное программное оборудование (расходы на среду)?

- Каково соотношение между продуктивным тестированием и (непродуктивной) интеграцией?
- Каковы затраты на версию N (являющиеся основанием для планирования затрат на версию N+1)?

10.1.2 Эволюционирующие декомпозиции работ

Эволюционирующие WBS организуют элементы планирования вокруг схемы процесса, а не вокруг схемы продукта. Такой подход позволяет лучше подстроиться под ожидаемые изменения в постоянно совершенствующемся плане и допускает изменение уровня качества планирования в прямом направлении. Основной рекомендацией относительно WBS является организация иерархии следующим образом:

- Элементами WBS первого уровня являются рабочие процессы (управление проектом, создание рабочей среды, управление требованиями, проектирование, реализация, оценка и внедрение). Эти элементы обычно закрепляются за одной командой (см. главу 11) и формируют «анатомию» проекта, которая используется в целях планирования и сравнения с другими проектами.
- Элементы второго уровня определяются для каждой стадии жизненного цикла (начальная стадия, уточнение, конструирование и ввод в действие). Эти элементы позволяют естественным образом повышать точность плана параллельно с повышением уровня понимания требований и архитектуры, а также таящихся в них рисков.
- Элементы третьего уровня определяются для выделения видов деятельности, в результате которых производятся рабочие продукты каждой стадии. Эти элементы могут либо образовывать самый нижний уровень в иерархии, который позволяет вычислить стоимость отдельного вида рабочих продуктов для данной стадии, либо разбиваться дальше на несколько видов деятельности более низких уровней, которые, взятые вместе, обеспечивают получение одного вида рабочих продуктов.

Стандартная WBS, соответствующая схеме процесса (стадии, рабочие процессы и рабочие продукты), показана на рис. 10.2. Эта рекомендуемая структура является примером того, каким образом элементы процесса могут быть сведены в план. Она предлагает схему для оценки затрат и сроков по каждому элементу, их распределения по организации, выполняющей проект, и отслеживания расходов.

Приведенная структура должна рассматриваться как отправная точка. Ее необходимо подогнать под особенности проекта по многим направлениям.

- Масштаб. Более масштабные проекты будут иметь больше уровней и подструктур.

- Организационная структура. Проекты, где задействованы субподрядчики или участвует множество различных организаций, могут иметь ограничения, которые приведут к необходимости иного распределения WBS.
- Объем разработок на заказ. В зависимости от характера проекта в рабочих процессах управления требованиями, проектирования и реализации внимание может уделяться разным аспектам. Проект реинжиниринга бизнес-процессов, основанный по большей части на уже существующих компонентах, должен иметь большую глубину детализации требований и не слишком углубляться в проектирование и реализацию. Разработка единственного в своем роде технического приложения, выполняемая полностью на заказ, может потребовать действительно глубокой проработки в части проектирования и реализации, для того чтобы справиться с рисками первого поколения новых компонентов.
- Бизнес-контекст. Проекты, выполняемые на контрактной основе, требуют более совершенного управления и оценки. Проекты, в которых разрабатываются коммерческие продукты для продажи широкому кругу потребителей, могут потребовать более совершенных структур для внедрения. Приложение, внедряемое в единственном месте, может обладать как совершенно тривиальным элементом внедрения (например, в случае разработки бизнес-приложения внутри организации), так и хорошо проработанным (например, при переходе от критически важной унаследованной системы с обеспечением параллельной эксплуатации для достижения нулевого времени простоя).
- Предшествующий опыт. Очень немногие проекты начинаются с чистого листа. Большинство из них разрабатывается либо как новые поколения унаследованных систем (с устоявшейся WBS), либо в контексте существующих организационных стандартов (с предопределенным построением WBS). Важно подстроиться под эти ограничения для гарантии, что новый проект сможет воспользоваться имеющимся опытом и достигнутым уровнем производительности.

WBS разбивает содержимое проекта на части и ставит их в соответствие жизненному циклу, бюджету и персоналу. Рассмотрение WBS позволяет уточнить важные атрибуты, приоритеты и структуру плана проекта. Выполняя оценку проектов и аудит управления проектами по созданию ПО на протяжении последних нескольких лет, автор обнаружил, что WBS является наиболее ценным источником объективной информации о плане проекта. В то время как план разработки ПО и бизнес-план создают контекст для рассмотрения, WBS и соответствующие бюджеты, распределяемые по элементам, являются наиболее осмысленными показателями подхода, приоритетов и приемов управления.

A Управление проектом
- AA Управление начальной стадией
  - AAA Разработка бизнес-плана
  - AAB Спецификации версии стадии уточнения
  - AAC Определение основ WBS стадии уточнения
  - AAD План разработки ПО
  - AAE Контроль и оценка состояния проекта на начальной стадии
- AB Управление стадией уточнения
  - ABA Спецификации версии стадии конструирования
  - ABB Определение основ WBS стадии конструирования
  - ABC Контроль и оценка состояния проекта на стадии уточнения
- AC Управление стадией конструирования
  - ACA Планирование стадии внедрения
  - ACB Определение основ WBS стадии внедрения
  - ACC Контроль и оценка состояния проекта на стадии конструирования
- AD Управление стадией ввода в действие
  - ADA Планирование следующего поколения
  - ADB Контроль и оценка состояния проекта на стадии ввода в действие

B Создание рабочей среды
- BA Спецификации среды на начальной стадии
- BB Базовая среда на стадии уточнения
  - BBA Инсталляция и администрирование среды разработки
  - BBB Интеграция среды разработки и настройка инструментов
  - BBC Формирование базы данных запросов на изменение
- BC Сопровождение среды на стадии конструирования
  - BCA Инсталляция и администрирование среды разработки
  - BCB Сопровождение базы данных запросов на изменение
- BD Сопровождение среды на стадии ввода в действие
  - BDA Сопровождение и администрирование среды разработки
  - BDB Сопровождение базы данных запросов на изменение
  - BDC Формирование и передача среды сопровождения

C Управление требованиями
- CA Разработка требований на начальной стадии
  - CAA Описание концепции
  - CAB Моделирование вариантов использования
- CB Определение основных требований на стадии уточнения
  - CBA Определение основ концепции
  - CBB Моделирование основных вариантов использования
- CC Сопровождение требований на стадии конструирования
- CD Сопровождение требований на стадии ввода в действие

Рис. 10.2. *Стандартная декомпозиция работ*

D Проектирование
- DA Создание архитектурных прототипов на начальной стадии
- DB Определение базовой архитектуры на стадии уточнения
  - DBA Моделирование проекта архитектуры
  - DBB Планирование и проведение демонстрации проекта
  - DBC Описание архитектуры ПО
- DC Моделирование проекта на стадии конструирования
  - DCA Сопровождение модели проекта архитектуры
  - DCB Моделирование компонентов
- DD Сопровождение проекта на стадии ввода в действие

E Реализация
- EA Создание прототипов компонентов на начальной стадии
- EB Реализация компонентов на стадии уточнения
  - EBA Интеграция для демонстрации кодирования критичных компонентов
- EC Реализация компонентов на стадии конструирования
  - ECA Кодирование компонентов для начальных версий и их автономное тестирование
  - ECB Кодирование компонентов для альфа-версии и их автономное тестирование
  - ECC Кодирование компонентов для бета-версии и их автономное тестирование
  - ECD Сопровождение компонентов
- ED Сопровождение компонентов на стадии ввода в действие

F Оценка
- FA Планирование оценок на начальной стадии
- FB Оценки на стадии уточнения
  - FBA Моделирование тестов
  - FBB Реализация сценариев тестирования архитектуры
  - FBC Оценка демонстрации и описания версии
- FC Оценки на стадии конструирования
  - FCA Оценка начальной версии и описание версии
  - FCB Оценка альфа-версии и описание версии
  - FCC Оценка бета-версии и описание версии
- FD Оценки на стадии ввода в действие
  - FDA Оценка версии продукта и описание версии

G Внедрение
- GA Планирование внедрения на начальной стадии
- GB Планирование внедрения на стадии уточнения
- GC Внедрение на стадии конструирования
  - GCA Определение основ руководства пользователя
- GD Внедрение на стадии ввода в действие
  - GDA Передача продукта пользователю

Рис. 10.2. *Стандартная декомпозиция работ (продолжение)*

Другой важной особенностью хорошей WBS является то, что точность планирования, присущая каждому элементу, согласуется с текущей стадией жизненного цикла и состоянием проекта. Эта идея представлена на рис. 10.3. Одной из главных причин, по которой WBS организована именно так, является необходимость в учете изменения элементов плана от общего планирования (приблизительные бюджеты, которые приводятся в качестве оценки для последующего уточнения, а не расписываются подробно по частям) до сетки окончательно спланированных видов деятельности (с точно определенным бюджетом и постоянной оценкой реального состояния по сравнению с планируемыми затратами).

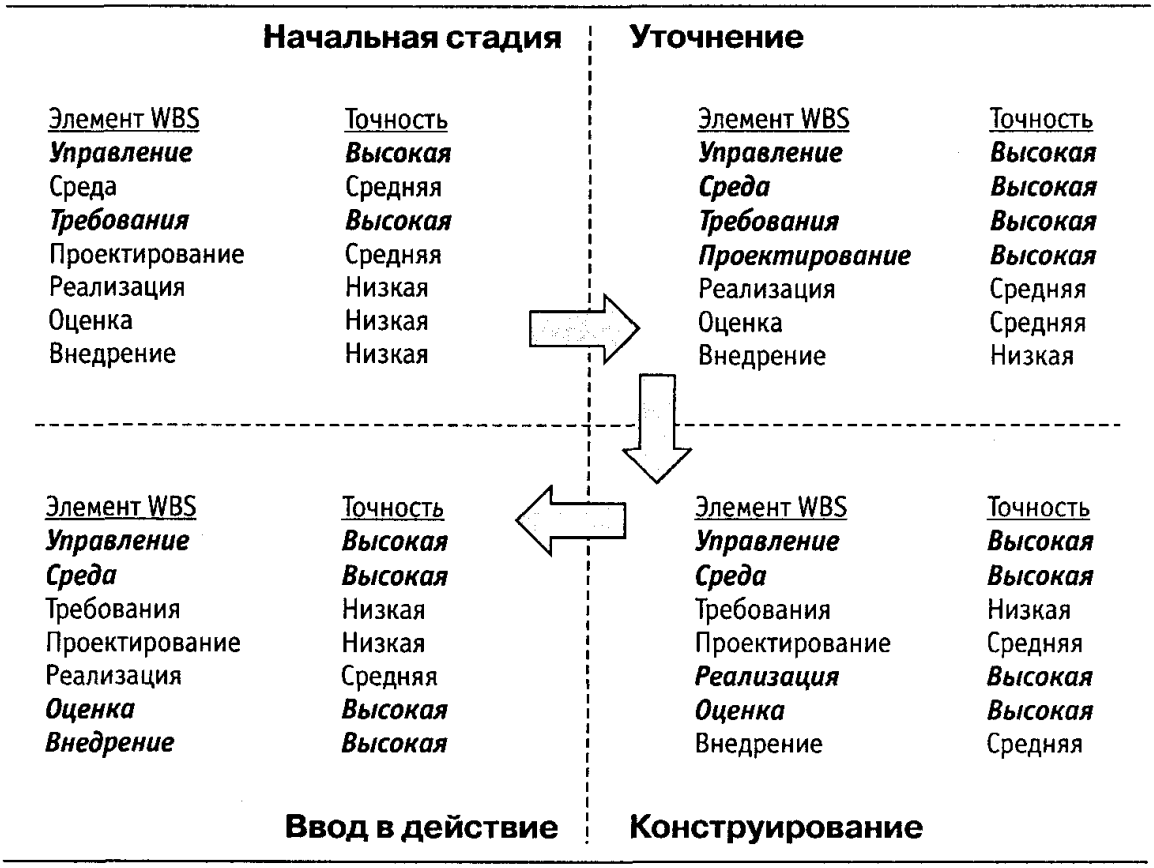


Рис. 10.3. *Эволюция точности планирования в WBS на протяжении жизненного цикла*

10.2 МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПЛАНИРОВАНИЮ

Проекты по созданию ПО охватывают самый широкий спектр областей применения. Было бы очень ценным, хотя и рискованным, сделать рекомендации по планированию независимыми от контекста проекта. Это полезно, поскольку многим из тех, кто занимает руководящие посты, требуется отправная точка, на основе которой они могли бы учесть особенности, присущие данному проекту. Они понимают, что начальные руководства по планированию содержат экспертные оценки и опыт

многих людей. Поэтому такие указания рассматриваются как заслуживающая доверия основа для оценок и внушают определенную уверенность заинтересованным сторонам.

Независимые от проекта советы по планированию также весьма рискованны. Существует риск, что указания будут применяться буквально, без адаптации к обстоятельствам конкретного проекта. Слепое следование чьим бы то ни было независимым от проекта рекомендациям по планированию является признаком некомпетентности команды управления. Кроме того, существует риск неверной интерпретации. Изменчивость параметров проекта, его бизнес-контекста, организационной культуры и процессов, характерных для различных проектов, усугубляют возможность допущения ошибок, которые потенциально могут оказать большое влияние. В этой книге я старался подобрать соответствующий контекст с тем, чтобы при интерпретации можно было избежать подобных ошибок. Для того чтобы смягчить «проектно-независимые» дискуссии, в приложении D подробно представлен реальный проект. Это хороший пример проекта, который на 90% руководствуется указаниями по независимому от проекта планированию, приведенными в книге. Он также содержит примеры и обоснования нескольких незначительных отклонений от этих указаний.

Приступая к созданию или оценке плана, следует принять во внимание два простых указания по планированию. Первое указание (см. таблицу 10.1) дает предварительное описание стандартного распределения затрат между элементами WBS первого уровня. Второе указание (см. таблицу 10.2) дает предварительное описание распределения усилий и сроков между стадиями жизненного цикла. При наличии начальной оценки стоимости всего проекта и этих двух таблиц структура персонала, распределение людских ресурсов по различным командам, график выполнения проекта верхнего уровня и начальная WBS с бюджетом и сроками выполнения отдельных заданий оказываются очевидными. Эта разновидность разработки плана сверху вниз является полезным опытом по планированию, который обеспечивает основу для его дальнейшего уточнения.

Таблица 10.1.
Стандартные бюджеты WBS

Элемент WBS первого уровня	Стандартный бюджет
Управление проектом	10%
Создание рабочей среды	10%
Управление требованиями	10%
Проектирование	15%
Реализация	25%
Оценка	25%
Внедрение	5%
Итого	100%

Таблица 10.2.
Стандартное распределение усилий и сроков по стадиям

Область определения	Начальная стадия	Уточнение	Конструирование	Ввод в действие
Усилия	5%	20%	65%	10%
Сроки	10%	30%	50%	10%

Откуда взялись данные, приведенные в таблицах 10.1 и 10.2? К сожалению, не из банка данных хорошо документированных практических примеров многочисленных успешных проектов, в которых использовался современный процесс создания ПО. Эти данные взяты в основном из моего собственного опыта, включая участие в работах по определению стоимости ПО в течение последнего десятилетия; мне пришлось иметь дело с широким спектром проектов по созданию ПО, организаций, процессов и технологий.

В таблице 10.1 приводится стандартное распределение бюджетных средств по элементам WBS первого уровня. Поскольку эти значения изменяются от проекта к проекту, распределение дает хорошую точку отсчета для оценки плана, позволяя понять причины отклонений от этих контрольных значений. Важным моментом в данном случае является то, что это — распределение затрат, а не усилий. Во избежание ошибочных интерпретаций необходимо сделать два пояснения:

1. В этих цифрах учтена стоимость различных категорий трудозатрат. Например, управление проектом, управление требованиями и проектирование — это элементы, где обычно используется персонал с более высокими должностями и с более высокой оплатой. Если управление требованиями и проектирование совместно потребляют 25% бюджета (при использовании сотрудников со средней зарплатой $100/час), этой сумме может соответствовать всего лишь половина необходимого количества человеко-часов по сравнению с элементом оценки, который, хотя и потребляет те же 25% бюджета, но задействует персонал со средним заработком $50/час.

2. В элемент создания рабочей среды включена также стоимость программной и аппаратной составляющих, необходимых для поддержки автоматизации процесса и команд разработчиков.

В таблице 10.2 приводятся рекомендации по распределению усилий и сроков по стадиям жизненного цикла. Эти значения также могут меняться в широких пределах в зависимости от специфических особенностей приложения, тем не менее они дают среднее ожидаемое значение для целого спектра областей применения. Достигнуть непротиворечивости, используя именно эти конкретные значения, не так важно, как понять, почему ваш проект может иметь другие значения.

10.3 ПРОЦЕСС ОЦЕНКИ СРОКОВ И ЗАТРАТ

Планы проекта должны создаваться с учетом двух точек зрения. Первая – это подход сверху вниз. Он начинается одновременно с пониманием общих требований и ограничений, приводит к определению бюджета и сроков на макроуровне, затем разбивает эти элементы на бюджеты более низких уровней и на промежуточные контрольные точки. С этой точки зрения при планировании должна выполняться такая последовательность действий:

1. Менеджер проекта по созданию ПО (и другие менеджеры) вырабатывает общую оценку размера проекта, процесса, среды, персонала и требуемого качества.
2. Производится приблизительная оценка общих усилий и сроков с использованием модели оценки стоимости.
3. Менеджер проекта детализирует эту приблизительную оценку усилий на верхнем уровне WBS, используя рекомендации, аналогичные приведенным в таблице 10.1. Менеджер проекта детализирует также сроки, устанавливая даты основных контрольных точек, и распределяет необходимые усилия в соответствии со штатным расписанием, используя рекомендации, аналогичные приведенным в таблице 10.2. Таким образом получается план проекта верхнего уровня. Подобные приблизительные оценки обычно не учитывают многочисленные параметры, специфичные для данного проекта.
4. В этот момент на менеджеров отдельных частей проекта возлагается ответственность за разбиение каждого из элементов WBS на элементы более низких уровней, учитывающее их расположение на верхнем уровне, штатное расписание и даты основных контрольных точек в качестве ограничений.

Вторая точка зрения – прямо противоположный подход снизу вверх. Вы анализируете бюджеты и сроки микроуровня, а затем складываете эти элементы в бюджеты более высоких уровней и промежуточные контрольные точки. Такой подход позволяет определять и наполнять WBS, начиная с самых нижних уровней и двигаясь вверх. С этой точки зрения при планировании должна выполняться следующая последовательность действий:

1. Элементы WBS самого нижнего уровня прорабатываются в виде отдельных заданий, сроки и бюджеты для которых приблизительно оцениваются менеджерами, ответственными за данный элемент WBS. Такого рода приблизительные оценки обычно имеют тенденцию учитывать специфику данного проекта в преувеличенном виде.
2. Приблизительные оценки суммируются и объединяются в бюджеты и контрольные точки более высоких уровней. Из-за необъективности отдельных лиц, выполняющих оценку, приходится приводить их к общему знаменателю с тем, чтобы иметь непротиворечивую основу для переговоров.

3. Производится сравнение с бюджетами и контрольными сроками, разработанными сверху вниз. Определяются самые значительные расхождения и делаются уточнения для того, чтобы достигнуть общего согласования между оценками, выполненными сверху вниз и снизу вверх.

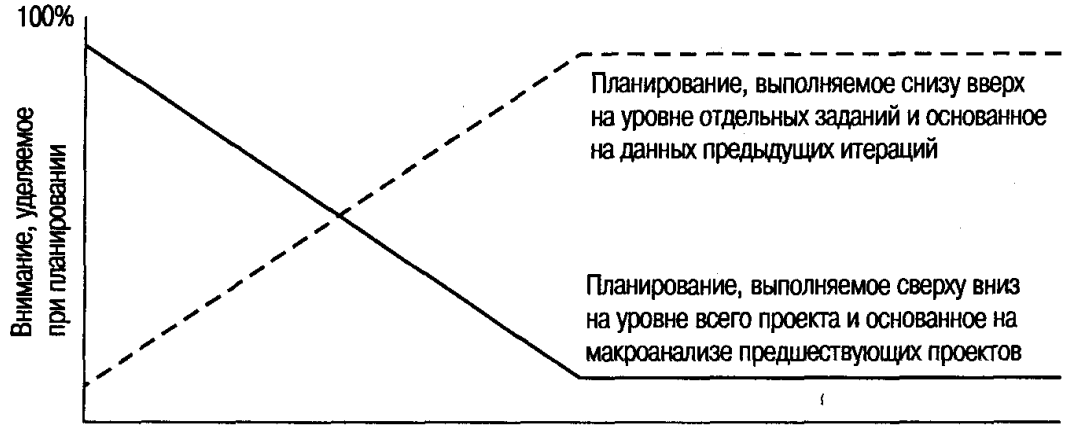
Определение сроков достижения контрольных точек или распределение бюджетов посредством оценок, выполняемых по принципу сверху вниз, имеет тенденцию к преувеличению субъективного мнения руководства проектом и обычно приводит к созданию чрезмерно оптимистичного плана. Оценки, выполняемые по принципу снизу вверх, обычно усиливают предвзятость исполнителей и приводят к созданию чрезмерно пессимистичного плана. Требуется еще одна итерация, которая бы использовала результаты, полученные с помощью одного подхода, для подтверждения и уточнения результатов, полученных с помощью другого подхода, тем самым формируется несколько уточняющихся версий плана. Такой процесс вселяет уверенность относительно правильности плана на всех уровнях управления.

Эти два подхода к составлению плана должны применяться вместе, взвешенно, на протяжении всего жизненного цикла. На стадии разработки будет доминировать подход сверху вниз, поскольку в этот момент обычно еще не существует ни достаточной глубины понимания, ни стабильности, присущей подробно разработанной последовательности заданий, необходимых для достоверного планирования по принципу снизу вверх. К стадии производства должно накопиться достаточное количество опыта планирования, что приводит к доминированию способа снизу вверх. С этого момента подход сверху вниз должен тщательно настраиваться в соответствии со специфичными для данного проекта параметрами, поэтому его следует использовать в большей степени как общую методику оценки. На рис. 10.4 показан баланс между различными способами планирования на протяжении жизненного цикла.

10.4 ПРОЦЕСС ПЛАНИРОВАНИЯ ИТЕРАЦИЙ

До настоящего момента наша дискуссия имела отношение исключительно к тем аспектам определения бюджетов и сроков, которые не зависят от области применения. Еще одно направление планирования касается определения реальной последовательности промежуточных результатов. Планирование содержания и сроков для основных контрольных точек и промежуточных итераций является, вероятно, одной из наиболее значимых форм общего плана по управлению рисками. План последовательных этапов создания весьма важен, поскольку всегда существуют поправки, вносимые в содержание и сроки этих этапов по мере того, как предположения, сделанные на ранних стадиях, превращаются в ясно осознаваемые обстоятельства проекта.

Общая последовательность создания и общие указания по числу итераций, необходимых для каждой стадии, обсуждаются ниже. *Итерация* обычно означает полную синхронизацию на протяжении выполнения проекта с хорошо организованной глобальной оценкой проекта в целом.



Стадия разработки		Стадия изготовления	
Начальная стадия	Уточнение	Конструирование	Ввод в действие
Итерации по определению реализуемости проекта	Итерации по созданию архитектуры	Итерации по реализации продукта	Версии продукта

Основное внимание, уделяемое на стадии разработки:
- Оценка заданий на макроуровне для рабочих продуктов стадии производства
- Оценка заданий на микроуровне для рабочих продуктов стадии разработки
- Согласование со всеми заинтересованными сторонами
- Укрупненный анализ расхождений между реальными и планируемыми затратами
- Переработка независимых от проекта указаний по проведению планирования сверху вниз в указания по проведению планирования, специфичного для данного проекта
- Определение и разработка WBS

Основное внимание, уделяемое на стадии производства:
- Оценка заданий на микроуровне для рабочих продуктов стадии производства
- Оценка заданий на макроуровне для сопровождения рабочих продуктов стадии разработки
- Согласование со всеми заинтересованными сторонами
- Тонкий анализ расхождений между реальными и планируемыми затратами

Рис. 10.4. *Баланс между различными видами планирования на протяжении жизненного цикла*

Остальные микроитерации, такие как ежемесячные, еженедельные или ежедневные работы, выполняются по мере движения к точкам синхронизации на уровне проекта.

- Итерации на начальной стадии. Ранние работы по созданию прототипов интегрируют основные компоненты предполагаемой архитектуры и обеспечивают реализуемую основу для детализации критичных вариантов использования системы. Такая основа включает в себя уже существующие компоненты, коммерческие компоненты и прототипы, сделанные на заказ, достаточные для того, чтобы продемонстрировать предполагаемую архитектуру и понимание требований, обеспечивающее создание достоверного бизнес-плана, общей концепции и плана разработки ПО. Крупномасштабные,

выполняемые на заказ разработки могут потребовать двух итераций для создания приемлемого прототипа, однако для большинства проектов оказывается вполне достаточно одной итерации.

- Итерации на стадии уточнения. Эти итерации приводят к созданию архитектуры, включая полную основу и инфраструктуру для функционирования системы. По завершении «архитектурной» итерации необходимо продемонстрировать некоторые критичные варианты использования. Среди них: (1) инициализация архитектуры, (2) включение сценария для того, чтобы пропустить через систему поток данных для обработки при самых плохих условиях (например, выполнение транзакций или сценария при пиковой нагрузке) и (3) введение сценария для контроля за прохождением через систему потока управления при самых плохих условиях (например, вариантов использования для восстановления после сбоев). В большинстве проектов для получения приемлемой базовой архитектуры необходимо запланировать две итерации. Для архитектуры, не имеющей прецедентов, может потребоваться несколько дополнительных итераций, в то время как проекты, основанные на устоявшейся схеме архитектуры, смогут, вероятно, обойтись одной итерацией.

- Итерации на стадии конструирования. Для большинства проектов на стадии конструирования требуются по крайней мере две основные итерации: альфа-версия и бета-версия. В альфа-версию должны быть включены функциональные возможности для всех критичных вариантов использования. Обычно они представляют лишь около 70% от общего объема продукта и выполняются с более низким уровнем качества (производительность и надежность), чем тот, что ожидается в конечном продукте. Бета-версия, как правило, обеспечивает 95% всех возможностей продукта, и в ней достигаются некоторые из наиболее важных характеристик качества. Однако обычно требуется завершить реализацию еще нескольких возможностей, а также улучшить стабильность и производительность для приемки окончательной версии продукта. Хотя для большинства проектов достаточно двух итераций на стадии конструирования, существует много причин, по которым следует добавить одну или две дополнительные итерации для контроля за рисками или оптимизации использования ресурсов.

- Итерации на стадии ввода в действие. Большинство проектов требует наличия единственной итерации по превращению бета-версии в конечный продукт. Опять-таки может понадобиться огромное количество неформальных маленьких итераций для исправления всех дефектов, для использования в бета-версии результатов, полученных по обратной связи с пользователями, и для повышения производительности. Однако по причине накладных расходов, связанных с полномасштабной передачей продукта пользовательскому сообществу, многие проекты умудряются обходиться одной итерацией между бета-версией и окончательным выпуском продукта.

Общая рекомендация такова, что для большинства проектов потребуется от четырех до девяти итераций. Типичный проект будет состоять из шести итераций и иметь следующий вид:

- Одна итерация на начальной стадии: архитектурный прототип
- Две итерации на стадии уточнения: архитектурный прототип и базовая архитектура
- Две итерации на стадии конструирования: альфа- и бета-версии
- Одна итерация на стадии ввода в действие: выпуск продукта

Проекты, имеющие много прецедентов, с заранее определенной архитектурой, либо очень небольшие проекты могут обходиться одной итерацией на начальной стадии, объединенной со стадией уточнения; в результате для выпуска продукта используется всего лишь четыре итерации. Очень большой проект либо проект, не имеющий прецедентов, с участием множества заинтересованных сторон может потребовать одной дополнительной итерации на начальной стадии и двух дополнительных итераций на стадии конструирования, что в сумме составит девять итераций. Связанные с этим накладные расходы могут оказаться вполне разумной платой за уверенность в надлежащем управлении рисками и в согласовании действий всех заинтересованных сторон.

10.5 ПРАКТИЧЕСКОЕ ПЛАНИРОВАНИЕ

Хорошее планирование в условиях итерационного процесса является более динамичным, тем не менее осуществлять его с достаточной точностью оказывается значительно проще. В процессе выполнения итерации N на любой стадии менеджер проекта по созданию ПО должен отслеживать и контролировать соответствие плану, созданному в процессе выполнения итерации N-1, и при этом планировать итерацию N+1. Высшее искусство управления проектом заключается в том, чтобы достигать всех согласований в планах текущей итерации и в планах очередной итерации, основываясь на объективных результатах текущей и предшествующей итераций. Такая концепция представляется — и является — трудноосуществимой на ранних стадиях или в проектах, прокладывающих путь к итерационной разработке. Но если механизм планирования отлажен, процесс становится удивительно простым по мере перехода к стадиям, в которых качественное планирование является залогом успеха.

Помимо плохой архитектуры и неправильного понимания требований, неадекватное планирование (и соответственно плохое управление) является одной из наиболее распространенных причин неудачного завершения проектов. Напротив, удача проекта может быть частично отнесена на счет хорошего планирования, управления требованиями и архитектуры. Конечным продуктам, связанным с этими аспектами (плану разработки ПО, спецификациям требований и документу с описанием архитектуры), не уделяется особого внимания. Для большинства успешных проектов они не являются очень важными после того, как были созданы.

Они редко используются исполнителями каждый день и не представляют интереса для конечного пользователя, а их фиксация на бумаге является лишь верхушкой айсберга с учетом деталей, лежащих в их основании.

Хотя планирующий документ не слишком полезен в качестве конечного изделия, сам акт планирования чрезвычайно важен для успеха всего проекта. Он представляет собой основу и стимул для принятия решений, гарантирует обеспечение всем необходимым всех заинтересованных сторон и исполнителей и преобразует субъективные, общие схемы процесса в объективные процессы. План проекта — это определение того, каким образом требования к проекту будут трансформированы в продукт с учетом экономических ограничений. Он должен быть реалистичным, современным, являться плодом коллективного творчества, понятным всем заинтересованным сторонам. И он должен использоваться.

Планы создаются не только для менеджеров. Чем более процесс планирования и его результаты открыты и наглядны, тем большее участие в них принимают члены команды. Плохие, скрытно выполненные планы приводят к трениям. Хорошие, открытые планы формируют культуру и способствуют коллективному творчеству.

Глава 11

Проектные организации и распределение обязанностей

Ключевые моменты

- Организационная структура определяет архитектуру команд исполнителей.
- Организации, вовлеченные в бизнес ПО, нуждаются в поддержке проектов посредством инфраструктуры, необходимой для использования общего процесса.
- Организации, работающие над проектом, должны четко распределять среди проектных команд рабочие продукты и обязанности, чтобы гарантировать баланс между глобальным (архитектура) и локальным (компонент) аспектами.
- Организация должна изменяться вместе с изменением WBS и аспектов жизненного цикла.

Бизнес ПО и команды, работающие над проектами, имеют различные мотивации. Бизнес ПО мотивируется отдачей от инвестиций, новизной, диверсификацией рынка и прибыльностью. Команды, работающие над проектами, мотивируются стоимостью, сроками и качеством конкретных продуктов.

Профессионалы в области ПО в обоих типах организаций мотивируются карьерным ростом, удовлетворением от работы и возможностью выбора. Эта тема хорошо освещена в «A Discipline for Software Engineering» [Humphrey, 1995].

В прошлом все внимание при организации уделялось (и справедливо) проекту, т.е. уровню, на котором ПО разрабатывалось и внедрялось. При выполнении проектов преследовались эгоистические интересы, в рамках проектов редко делались инвестиции в какие-либо технологии или услуги, если те не оказывали непосредственного влияния на стоимость, сроки или качество получаемого в результате продукта. В этой главе рекомендуются и описываются различные способы организации для ведения бизнеса и для выполнения проекта. Предварительное описание организационных иерархий является опасным предприятием, если учитывать

особенности конкретных организаций и людей. Мы обсудим общие роли, взаимоотношения и распределение обязанностей. Для любого данного проекта или направления бизнеса эти обсуждения следует рассматривать как отправную точку. Их адаптация к конкретной предметной области, масштабу, культуре и персоналиям для некоторой конкретной ситуации может привести к целому набору различных реализаций. Может оказаться подходящей, например, иная организация проекта или команды, с разделением или объединением представленных ролей. Как бы то ни было, наши рекомендации основаны на опыте успешных проектов и могли бы предложить подходящую схему для большинства организаций.

11.1 ОРГАНИЗАЦИИ РАЗЛИЧНЫХ ОТРАСЛЕЙ ПРОМЫШЛЕННОСТИ

На рис. 11.1 показано распределение ролей и обязанностей в обычной промышленной организации. Эта структура может адаптироваться к конкретным обстоятельствам.

Обычная организация имеет следующие основные особенности:

- Ответственность за описание процесса и его сопровождение специфична для конкретного направления бизнеса, где имеет смысл говорить об общности процессов. Например, процесс разработки ПО для авиации отличен от процесса, используемого для разработки офисных приложений.

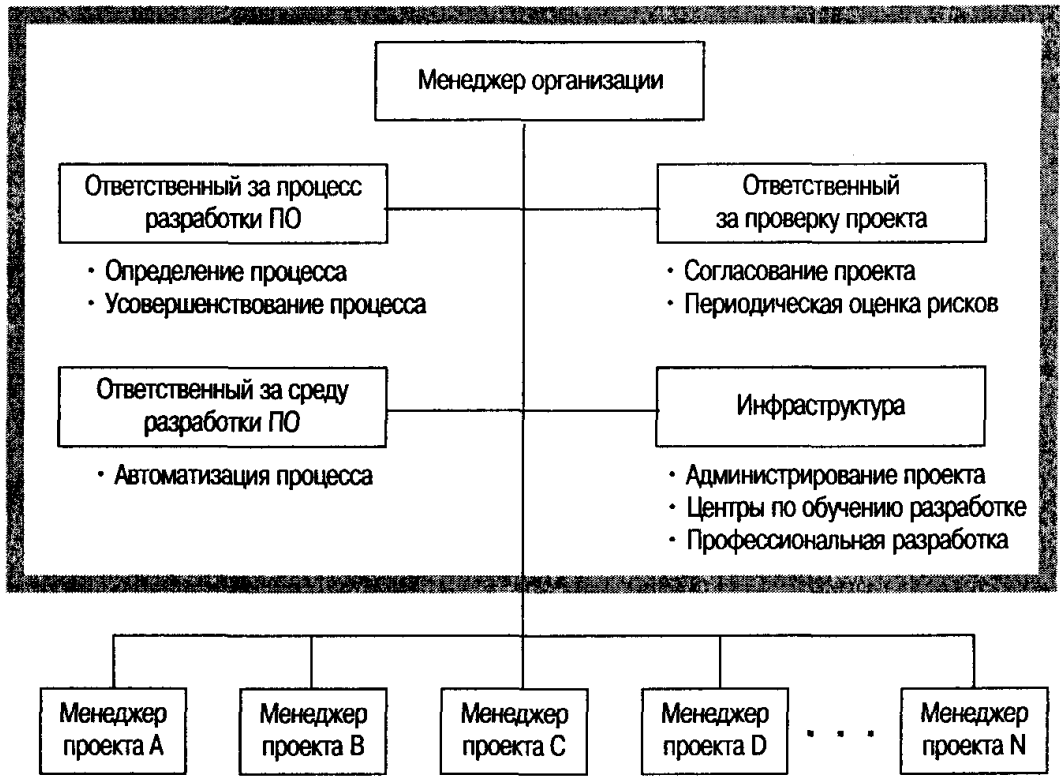


Рис. 11.1. *Стандартные роли в организации*

- Ответственность за автоматизацию процесса — это организационная роль, которая по важности равна роли определения процесса. Проекты достигают общности процессов прежде всего за счет поддержки общего инструментария.
- Организационные роли могут выполняться отдельным индивидуумом или несколькими различными командами, в зависимости от масштаба организации. Компании по производству ПО со штатом 20 человек может хватить одного человека для выполнения всех ролей, в то время как телекоммуникационной компании со штатом в 10 000 человек требуются сотни людей для создания эффективной организации по разработке ПО.

Ответственный за процесс разработки ПО

Ответственный за процесс разработки ПО (Software Engineering Process Authority, SEPA) обеспечивает обмен информацией и руководствами по процессу между практиками, занимающимися проектом. Это лицо отвечает перед руководителем организации за выполнение текущих оценок зрелости процесса в данной организации и за планирование дальнейшего его развития. SEPA должен оказывать помощь при инициировании проекта и периодически оценивать его процессы. Распространение лучшей практики при создании ПО может выполняться только в том случае, если SEPA понимает как само необходимое улучшение, так и особенности конкретного проекта. Он является ответственным и подотчетным за определение процесса и его сопровождение (модификацию, совершенствование, внедрение новых технологий). Функции SEPA могут выполняться отдельным лицом, руководителем организации либо целой командой представителей. SEPA должен действительно быть ответственным — компетентным и обладающим властью, — а не бессильным перед неэффективной бюрократией должностным лицом.

Ответственный за проверку проекта

Ответственный за проверку проекта (Project Review Authority, PRA) — отдельное лицо, отвечающее за то, чтобы проект по созданию ПО подчинялся всем организационным и экономическим правилам, практике и стандартам, касающимся ПО. Менеджер проекта по созданию ПО ответственен за соблюдение требований контракта или каких-то других стандартов, которым подчиняется проект, и подотчетен PRA. PRA рассматривает вопросы соответствия проекта как контрактным обязательствам, так и обязательствам, вытекающим из организационной политики проекта. Заказчик следит за требованиями контракта, контрольными точками, контрактными обязательствами, ежемесячным анализом состояния управления проектом, прогрессом, качеством, затратами, сроками и риском. PRA рассматривает поручения заказчика, а также следит за строгим соблюдением политики организации, выполнением финансовых условий и за другими рисками и аспектами.

Ответственный за среду разработки ПО

Ответственный за среду разработки ПО (Software Engineering Environment Authority, SEEA) отвечает за автоматизацию процесса, сопровождение стандартной среды, обучение пользованию средой и сопровождение в организации наработок, пригодных для повторного использования. Роль, которую выполняет SEEA, необходима для достижения высокого уровня отдачи от вложенных ресурсов для обычного процесса. Инструменты, методы и обучение могут эффективно амортизироваться на протяжении нескольких проектов только в том случае, если кто-то (например, SEEA) будет отвечать за сопровождение и администрирование стандартной среды. Во многих случаях среда может быть улучшена, изменена по требованию или модифицирована, однако существование готового на 80% решения для каждого проекта критично для утверждения процесса, используемого организацией, и для обеспечения хорошего ROI от инструментов.

Инфраструктура

Инфраструктура организации обеспечивает поддержку людских ресурсов, независимые от проектов исследования и разработки, а также другие важнейшие направления, связанные с созданием ПО. Она может изменяться в диапазоне от тривиальной до глубоко окопавшейся бюрократии. Типичными компонентами организационной инфраструктуры являются следующие:

- Администрирование проекта: система учета времени; контракты, цены, сроки и условия; интеграция в корпоративные информационные системы.
- Инновационные центры: поддержка инструментария, заявок и предложений, независимые исследования и разработки.
- Профессиональный рост: внутренний выездной лагерь для обучения, найм персонала, сопровождение базы данных по квалификации персонала, библиотека специальной литературы, публикации.

Сервисный центр организации обеспечивает стандартную среду, которая финансируется за счет основного бизнеса и поддерживается в качестве основных фондов для всех проектов, выполняемых в рамках организации. SEEA является компаньоном SEPA. SEPA отвечает за определение и усовершенствование процесса, а SEEA – за автоматизацию процесса.

Представляется важным, чтобы менеджеры организации рассматривали среду для разработки ПО так же, как технические средства, т.е. как оборудование, входящее в основные фонды. В большинстве мелких и незрелых организаций существует сопротивление такому подходу; в них специальная разработка процесса и инструментальное обеспечение относятся к прямым затратам проекта. В зрелых организациях, занимающихся созданием ПО, процесс и инструментальное обеспечение должны являться основными фондами организации так же, как это происходит в других инженерных дисциплинах. А раз так, то они должны

финансироваться из капитальных ресурсов. Финансовые модели могут предполагать включение этих затрат в накладные расходы или в общие и административные затраты либо в смету проекта, основанную на его выполнении. В сегодняшней индустрии ПО, характеризующейся устаревшими методами отчетности, финансированием обеспечения инструментами за счет проектов и методами лицензирования ПО, относительно небольшое количество организаций перешло на подобную модель капитальных вложений для своих программных сред. Такими организациями являются, как правило, достаточно зрелые, крупномасштабные организации-разработчики ПО, которые достигли стабильного определения процесса и установили долговременное сотрудничество с поставщиками инструментальных средств.

11.2 ПРОЕКТНЫЕ ОРГАНИЗАЦИИ

На рис. 11.2 показана стандартная структура проектной организации, а также распределение ролей и обязанностей на уровне проекта в целом. Эта структура может адаптироваться к размеру и особенностям конкретной проектной организации.

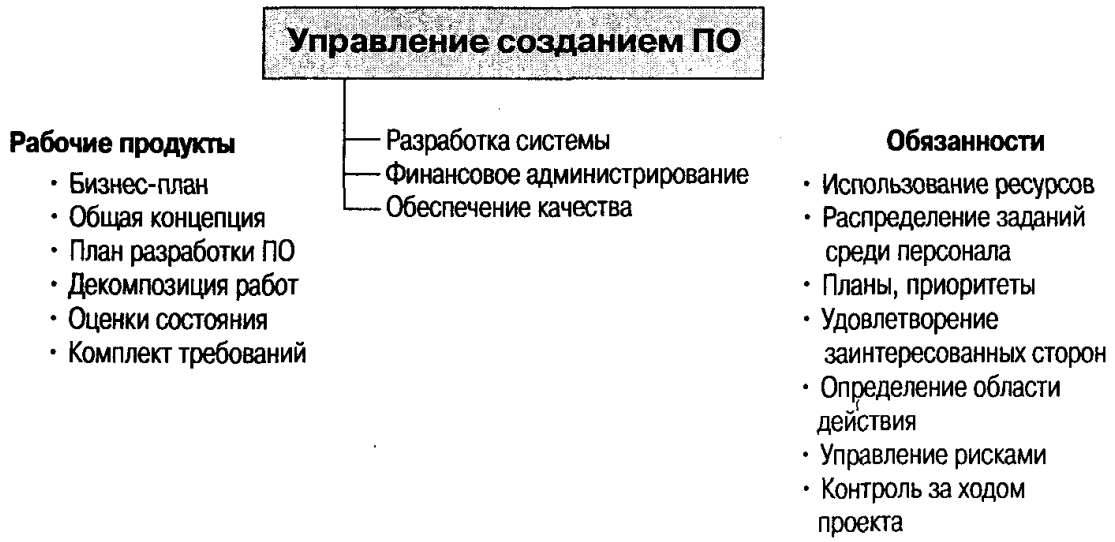
Основными особенностями стандартной организации являются следующие:

- Команда управления проектом является активным участником, ответственным за производство в той же степени, что и за управление. Управление проектом — это не спортивное «боление».
- Команда, разрабатывающая архитектуру, отвечает за реальные рабочие продукты и за интеграцию компонентов, а не только за выполнение руководящих функций.
- Команде по разработке поручается создание компонентов и сопровождение. Команда по оценке отделена от разработки. Такая структура благоприятствует независимому взгляду на качество и стимулирует осуществление тестирования и деятельности по оценке продукта параллельно с ведущейся разработкой.
- Качество — это всеобщая забота, оно проходит через все виды деятельности и контрольные точки. Каждая команда отвечает за определенный аспект качества.

Команда управления проектом

Большинство проектов страдает от избытка ограничений. Сроки, затраты, функциональные возможности и ожидаемое качество тесно взаимосвязаны и требуют постоянных согласований между многими заинтересованными сторонами, преследующими различные цели. Команда управления проектом отвечает за то, чтобы все заинтересованные стороны остались довольны полученными в проекте результатами. В этом отношении менеджеру проекта приходится каждый день беспокоиться о *балансе* интересов. На рис. 11.3 показано то, чему команда управления проектом должна уделять внимание на протяжении жизненного цикла.

Рис. 11.2. *Стандартная организация проекта и распределение обязанностей*



Внимание на протяжении жизненного цикла

Начальная стадия	Уточнение	Конструирование	Ввод в действие
Планирование стадии уточнения Создание команды Определение основ контракта Затраты на архитектуру	Планирование стадии конструирования Найм всего персонала Разрешение рисков Критерии приемки продукта Затраты на конструирование	Планирование стадии ввода в действие Оптимизация плана конструирования Управление рисками	Удовлетворение заказчика Закрытие контракта Поддержка продаж Планирование следующего поколения

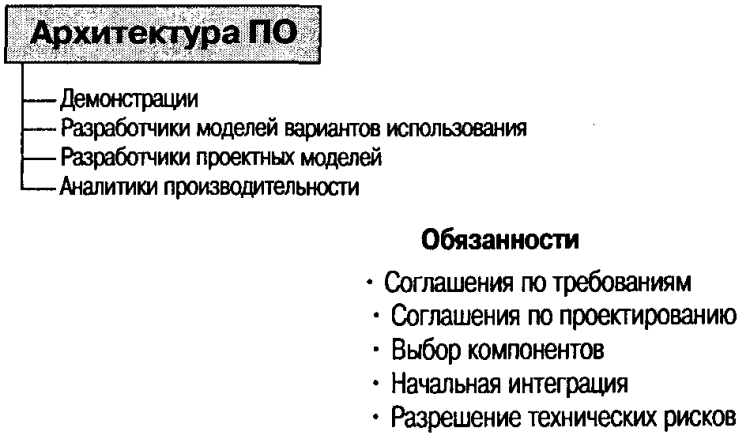
Рис. 11.3. *Виды деятельности команды управления проектом*

Команда управления проектом ответственна за планирование работ, выполнение плана и адаптацию плана к изменениям в требованиях или проекте. Для этого команда принимает на себя управление ресурсами и границами проекта, а также устанавливает рабочие приоритеты на протяжении всего жизненного цикла. На абстрактном уровне эта деятельность соответствует управлению ожиданиями всех заинтересованных сторон на протяжении жизненного цикла.

Команда управления проектом принимает на себя ответственность за все аспекты качества. В частности, она отвечает за поддержание такого баланса между этими аспектами, чтобы общее решение устраивало все заинтересованные стороны и было оптимальным для возможно большего их числа.

Команда по архитектуре ПО

Команда по архитектуре ПО отвечает за архитектуру. Эта ответственность включает в себя разработки, которые необходимы для определения полной спецификации материалов, требуемых ПО, и для принятия



Внимание на протяжении жизненного цикла

Начальная стадия	Уточнение	Конструирование	Ввод в действие
Создание архитектурного прототипа Соглашения по созданию/покупке Первичное определение сценария Определение критериев оценки архитектуры	Определение основ архитектуры Первичная демонстрация сценария Определение основ для соглашений относительно создания/покупки	Сопровождение архитектуры Разрешение проблем, касающихся нескольких компонентов Настройка производительности Повышение качества	Сопровождение архитектуры Разрешение проблем, касающихся нескольких компонентов Настройка производительности Повышение качества

Рис. 11.4. *Виды деятельности команды по архитектуре ПО*

решений относительно покупки/создания. При этом все компоненты, изготавливаемые на заказ, должны быть проработаны до такой степени, что затраты на изготовление/сборку стали абсолютно предсказуемыми. На рис. 11.4 показано то, чему команда по архитектуре ПО должна уделять внимание на протяжении всего жизненного цикла.

Квалификация команды по архитектуре ПО является критически важной для любого проекта. Она создает основу для взаимодействия между различными командами, достижения качества системы в целом и реализации приложений. При наличии хорошей команды по архитектуре можно добиться успеха со средней командой разработчиков. Если же архитектура слаба, то даже команда экспертов по разработке, состоящая из суперпрограммистов, вряд ли добьется успеха.

Для большинства проектов на начальной стадии и стадии уточнения будут доминировать две различные команды: команда управления проектом и команда по архитектуре ПО. (Даже это различие может оказаться размытым в зависимости от масштаба проекта.) При подготовке полномасштабной стадии производства существует тенденция привлекать команды по разработке и оценке ПО на вспомогательные роли. К тому времени, когда начинается стадия конструирования, архитектура переводится в режим сопровождения и должна поддерживаться минимальным уровнем усилий лишь для того, чтобы гарантировать непрерывную преемственность разработки.

Для достижения успеха команда по архитектуре должна обладать широкой компетенцией, включая следующее:

- Опыт в данной предметной области для создания приемлемых проектных решений (архитектурно значимых элементов проектной модели) и вариантов использования (архитектурно значимых элементов модели вариантов использования)
- Опыт в области технологии ПО для создания приемлемых процессов (связи параллельных потоков и потоков управления с проектными моделями, компонентами и моделями внедрения), компонентов (структуры комплекта реализации) и решений по внедрению (структуры комплекта внедрения)

Команда по архитектуре отвечает за качество системы в целом, которое включает в себя такие составляющие, как надежность, производительность и сопровождаемость. Эти составляющие охватывают множество компонентов и показывают, насколько хорошо они интегрированы для получения эффективного решения. В этом отношении команда по архитектуре определяет, каким образом должно решаться большинство проблем разработки, касающихся одновременно множества компонентов.

Разработка ПО

Команды по разработке компонентов

Рабочие продукты

- Комплект проектирования
- Комплект реализации
- Комплект внедрения

Обязанности

- Проектирование компонентов
- Реализация компонентов
- Автономное тестирование каждого компонента
- Сопровождение компонентов
- Документирование компонентов

Внимание на протяжении жизненного цикла

Начальная стадия	Уточнение	Конструирование	Ввод в действие
Поддержка создания прототипа Соглашения о создании/покупке	Разработка критичных компонентов Реализация и тестирование критичных компонентов Основы критичных компонентов	Проектирование компонентов Реализация компонентов Автономное тестирование каждого компонента Сопровождение компонентов	Сопровождение компонентов Документирование компонентов

Рис. 11.5. *Виды деятельности команды по разработке*

Команда по разработке ПО

На рис. 11.5 показано то, чему команда по разработке ПО должна уделять внимание на протяжении всего жизненного цикла.

Команда по разработке в большей степени, чем другие команды, зависит от специфики приложений. Вообще говоря, команда по разработке ПО распадается на несколько меньших команд, каждая из которых занимается отдельными группами компонентов, требующих определенного набора навыков. Типичные наборы навыков включают в себя следующее:

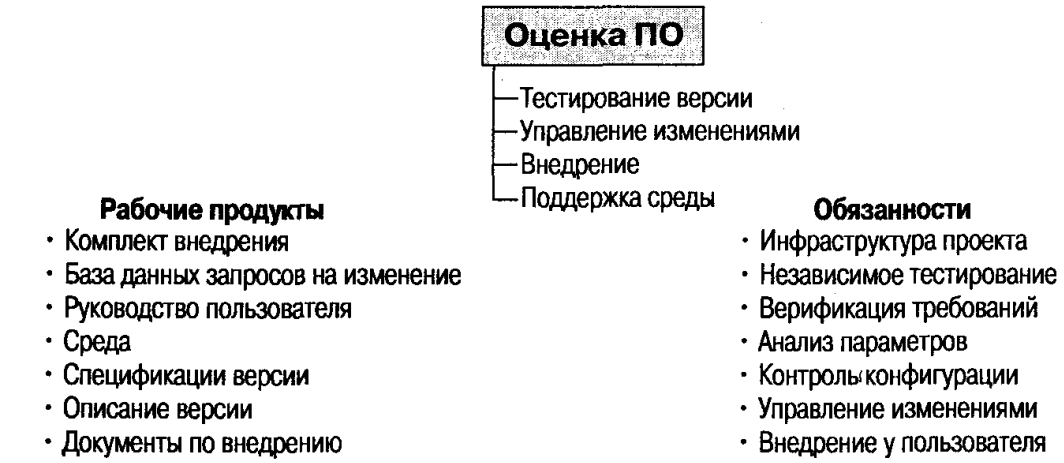
- Коммерческие компоненты: специалисты с детальным знанием коммерческих компонентов, занимающих центральное место в архитектуре системы
- Базы данных: специалисты с опытом организации, хранения и поиска данных
- Графические интерфейсы пользователя: специалисты с опытом организации вывода, представления данных и работы пользователя, необходимых для поддержки пользовательского интерфейса
- Операционные системы и сети: специалисты с опытом реализации множества программных объектов в сети аппаратных ресурсов, включая все типичные проблемы контроля, связанные с инициализацией, синхронизацией, разделением ресурсов, управлением пространством имен, перенастройкой, окончанием и межобъектными взаимодействиями
- Приложения в конкретных областях: специалисты в области создания алгоритмов, работы приложений или бизнес-правил, присущих данной системе

Команда по разработке ПО отвечает за качество отдельных компонентов, включая их разработку, тестирование и сопровождение. Тесты для компонентов следует организовывать как самодокументируемое многократно используемое ПО, которое будет рассматриваться так же, как исходный код других рабочих компонентов, сопровождаться естественным образом и будет доступно для автоматического регрессионного тестирования. В задачу команды по разработке входит решение любых локальных для компонентов проблем, касающихся разработки и реализации.

Команда по оценке ПО

На рис. 11.6 показано то, чему команда по оценке ПО должна уделять внимание на протяжении жизненного цикла проекта.

Существуют две причины использования независимой команды для оценки ПО. Первая касается гарантий независимости оценки качества. Этот часто обсуждаемый подход имеет свои «за» (например, гарантии того, что предубеждения авторов, коими являются разработчики, не исказят качество оценки) и «против» (например, освобождение команды по разработке ПО от ответственности за качество — до некоторой степени). Более серьезной причиной применения независимой команды тестирования является возможность осуществления различных видов



Внимание на протяжении жизненного цикла

Начальная стадия	Уточнение	Конструирование	Ввод в действие
Планирование инфраструктуры Создание прототипа основного сценария	Основы инфраструктуры Тестирование версии архитектуры Управление изменениями Начальное руководство пользователя	Обновление инфраструктуры Тестирование версии Управление изменениями Основы руководства пользователя Верификация требований	Сопровождение инфраструктуры Определение основ версии Управление изменениями Внедрение у пользователя Верификация требований

Рис. 11.6. *Виды деятельности команды по оценке*

деятельности одновременно. За счет выполнения разработки ПО и подготовки к тестированию, осуществляемой параллельно, можно добиться сокращения сроков. Управление изменениями, планирование тестов и создание сценария тестирования могут проводиться параллельно с проектированием и разработкой.

В современном процессе следует применять тестирование, ориентированное на варианты использования или тестирование отдельных возможностей (которое может охватывать несколько компонентов), организованное как последовательность этапов и автоматизированное посредством двух видов рабочих продуктов:

1. Спецификаций версии (план и критерии оценки версии)
2. Описания версии (результаты версии)

Каждая версия может охватывать несколько (возможно, незавершенных) компонентов, поскольку интеграция выполняется в течение продолжительного времени. В критериях оценки будет документировано то, что заказчик может ожидать при достижении основной контрольной точки, а описания версии смогут подтвердить результаты тестов. Заключительные итерации будут в общем эквивалентны приемочному тестированию;

они имеют уровни детализации, аналогичные уровням детализации традиционных отчетов, процедур и планов ПО. Эти материалы из очень кратких, абстрактных версий на ранних итерациях превращаются в более подробные и более строгие документы с проработанными деталями и обсуждениями подотчетных вопросов в более поздних версиях. Даже для тестирования вариантов использования тестовые компоненты должны разрабатываться способом, аналогичным способам разработки вариантов тестирования компонентов. Например, вместо того чтобы составлять документы по процедуре тестирования, в рамках проекта следует генерировать самодокументируемые сценарии тестирования, которые сами по себе являются программным обеспечением. Такие сценарии могут подвергаться внесению изменений точно так же, как другое ПО, и всегда должны содержаться в виде последней версии, пригодной для автоматизированного регрессионного тестирования.

Некоторые тесты компонентов могут быть возведены в ранг критериев оценки с включением их результатов в документацию с описанием версии. Многие компоненты могут проходить лишь неформальное тестирование, которое выполняется командой разработчиков и результаты которого будут храниться только внутри тестового ПО, созданного разработчиками. В таком случае формальное тестирование для многих компонентов будет отнесено к критериям оценки более высокого уровня (обычно это сценарии, ориентированные на определенную возможность или процесс) и к соответствующим описаниям версии. Компоненты не одинаковы: некоторые из них заслуживают формального покомпонентного тестирования для проверки соответствия требованиям, в то время как другие лучше тестируются в контексте какой-либо возможности. Решение этого вопроса следует оставить на усмотрение команды по оценке.

Команда по оценке отвечает за качество основных версий с учетом требований и ожиданий заказчика. Таким образом, команда по оценке ответственна за решение всех вопросов качества, влияющих на ожидания заказчика, независимо от того, отражены или нет эти ожидания в требованиях.

11.3 ЭВОЛЮЦИЯ ОРГАНИЗАЦИЙ

Проектная организация отражает архитектуру команды и должна изменяться в соответствии с планом проекта, содержащимся в декомпозиции работ. На рис. 11.7 показано, как центр тяжести команды сдвигается на протяжении жизненного цикла, когда около 50% сотрудников оказываются занятыми какой-то одной группой видов деятельности на каждой стадии.

На каждой стадии основное внимание уделяется определенным группам видов деятельности, а именно:

- Команда начальной стадии: основное внимание уделяется планированию, при этом со стороны других команд обеспечивается поддержка, достаточная, чтобы гарантировать консенсус планов со всех точек зрения.

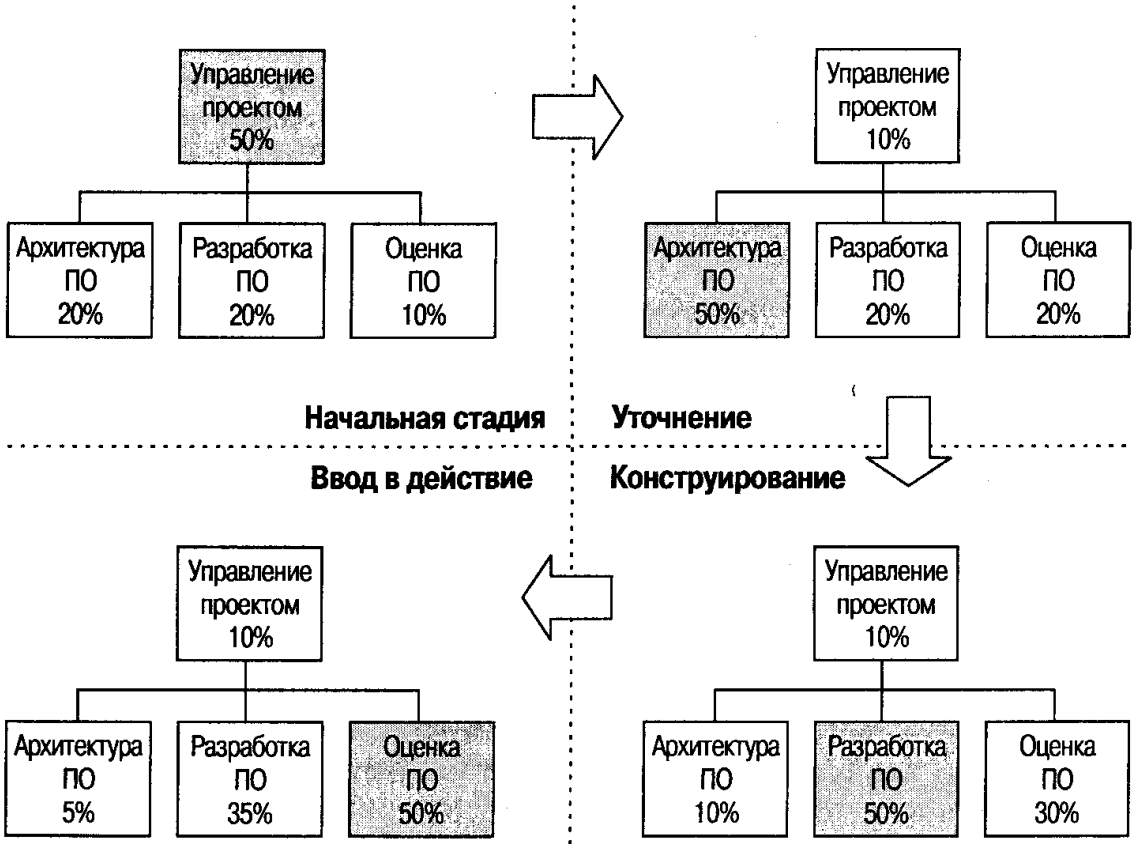


Рис. 11.7. *Эволюция команды, работающей над проектом по созданию ПО, на протяжении всего жизненного цикла*

- Команда стадии уточнения: основное внимание уделяется архитектуре, основные силы проекта собираются в команде по архитектуре, которая поддерживается командами по разработке ПО и по оценке ПО в степени, необходимой для получения стабильной базовой архитектуры.
- Команда стадии конструирования: сбалансированная команда, в которой большая часть деятельности приходится на команды разработки ПО и оценки ПО.
- Команда стадии ввода в действие: главное внимание уделяется заказчику, основным видом деятельности является внедрение, которым движет обратная связь с пользователями системы.

Не менее важна проработка всех деталей для команд, обязанностей и заданий более низкого уровня, однако она может осуществляться не ранее стабилизации всех деталей планирования в WBS. Преждевременное определение детальной структуры команды более низкого уровня может привести к серьезному снижению эффективности работ.

Глава 12

Автоматизация процесса

Ключевые моменты

- Среда разработки должна быть одним из важнейших рабочих продуктов процесса.
- Автоматизация процесса, в частности управление изменениями, важна для итерационного процесса. Если внесение изменений окажется дорогим, организация-разработчик будет противиться этому.
- «Круговая» разработка и интегрированные среды облегчают внесение изменений в рабочие продукты.
- Автоматизация измерения параметров критична для эффективного контроля над проектом.
- Всем внешним заинтересованным сторонам необходим доступ к ресурсам среды для взаимодействия с командой разработчиков и внесения в процесс дополнительных усовершенствований.

Многие организации-разработчики ПО сосредоточивают свое внимание на совершенствовании процесса для повышения степени предсказуемости управления проектом и положения дел в тех направлениях бизнеса, которыми они занимаются (это имеет отношение к качеству продукта, времени выхода на рынок, отдаче от инвестиций и производительности). Для того чтобы современные проекты по созданию ПО могли приносить прибыль, требуются не только определение процесса и его адаптация, но и значительный уровень автоматизации процесса.

Потребность в автоматизации возрастает с ростом масштаба работ. Точно так же, как процесс строительства будет изменяться в зависимости от того, строится ли домик для куклы, дом на одну семью или небоскреб, процесс создания ПО варьируется в широком диапазоне — от небольших заданий для одного человека, описание которых занимает одну страничку, до крупномасштабных, охватывающих несколько организаций, имеющих катастрофическую цену неудачи. Методы, обучение, временные рамки, критерии приемки и уровень автоматизации весьма различны для разных полюсов этого спектра.

Многие организации, занимающиеся созданием ПО, сталкиваются с необходимостью интеграции своей собственной среды и инфраструктуры в процесс создания ПО. Это обычно приводит к выбору более или менее несовместимых инструментов, которые имеют различные хранилища информации, поставляются разными производителями, работают на различных платформах, используют разную терминологию и основываются на различных подходах к процессу. Интеграция такой инфраструктуры оказывается гораздо проблематичнее, чем можно ожидать.

Автоматизация процесса разработки и создание инфраструктуры для поддержки различных рабочих процессов в рамках проекта являются важными видами деятельности на стадии жизненного цикла, посвященной разработке. Они включают в себя выбор инструментов, изготовление инструментов на заказ и автоматизацию процесса, т.е. то, что необходимо для выполнения работ в соответствии с планом создания и с приемлемой эффективностью. Превращение среды разработки в среду сопровождения также важно для долговременных проектов по разработке ПО.

Плохо то, что заинтересованные стороны проекта редко рассматривают среду как объект первостепенной важности, необходимый для длительного сопровождения продукта. Среда, обеспечивающая автоматизацию процесса, является весьма значимой составляющей, критичной для определения полной стоимости разрабатываемой системы. WBS верхнего уровня, рекомендованная в главе 10, рассматривает создание среды разработки как один из основных рабочих процессов.

В разделе 3.2 представлены три уровня процесса. Для эффективного выполнения каждого уровня требуется определенная степень автоматизации процесса:

1. *Метапроцесс*: политика, приемы и практика, которые присущи некоей организации при ведении интенсивного бизнеса, связанного с программным обеспечением. Автоматизация на этом уровне называется *инфраструктурой* (infrastructure). Инфраструктура – это перечень выбранных инструментов, шаблоны рабочих продуктов, методические рекомендации по микро- и макропроцессу, репозиторий проекта, база данных для хранения информации о квалификации персонала, библиотека примеров планов и результатов прошлых проектов.

2. *Макропроцесс*: политика, приемы и практика, которые присущи некоему проекту по созданию законченного ПО с учетом ограничений по стоимости, срокам и качеству. Автоматизация такого процесса называется средой (environment). Среда – это определенный набор инструментов для создания определенного комплекта рабочих продуктов в соответствии с определенным планом проекта.

3. *Микропроцесс*: политика, приемы и практика, которые присущи команде разработчиков некоего проекта и направлены на создание

рабочих продуктов в процессе разработки ПО. Автоматизация при создании каких-либо рабочих продуктов в общем случае называется *инструментом* (tool). Инструменты обычно включают в себя управление требованиями, визуальное моделирование, компиляторы, редакторы, отладчики, управление изменениями, автоматизацию измерения параметров, автоматизацию документирования, автоматизацию тестирования, оценку затрат и автоматизацию других рабочих процессов.

Поскольку основное внимание при автоматизации процесса направлено на среду проекта в целом, важными предпосылками для нее являются общее состояние инфраструктуры организации, выполняющей проект, и инструменты как строительный материал.

12.1 ИНСТРУМЕНТЫ: «КИРПИЧИКИ» АВТОМАТИЗАЦИИ

Существует множество инструментов, позволяющих автоматизировать процесс разработки ПО. В этом разделе дается обзор ядра среды, необходимого для поддержки процесса. В нем представлены наиболее важные универсальные инструменты, которые необходимы при выполнении различных проектов и хорошо соотносятся со схемой процесса. (Существует множество других инструментов и средств автоматизации процесса, не включенных в этот обзор.) Большинство из основных инструментов разработки ПО точно соответствует одному из рабочих процессов, как показано на рис. 12.1.

У каждого из рабочих процессов имеются свои, особенные требования к автоматизации. В одних случаях нужно генерировать некие рабочие продукты, в других — просто вести учет. Ниже обсуждаются некоторые важные вопросы, касающиеся каждого из рабочих процессов.

Управление проектом

Существует много возможностей для автоматизации процедуры планирования проекта и различных видов деятельности по контролю в рамках управления проектом. Инструменты для оценки затрат и WBS-инструменты оказываются полезными для генерации рабочих продуктов по планированию. Для управления в соответствии с планом лучше использовать такие инструменты для управления и такую «панель управления» проекта, которые поддерживают текущую оценку состояния проекта в онлайновом режиме. Такой подход к автоматизации может значительно улучшить понимание множества параметров и правил отчетности (см. главу 13). Общая концепция «панели управления» проекта по созданию ПО обсуждается в разделе 13.6.

Среда

В современном итерационном процессе разработки большое значение имеют управление конфигурацией и контроль над версиями. Тот подход к работе с параметрами, который рекомендован в главе 13, оказывается

Рис. 12.1. *Типичные компоненты автоматизации и инструменты, которые поддерживают различные рабочие процессы*

зависимым от изменений в рабочих продуктах. Некоторые аспекты автоматизации управления изменениями, которые должны поддерживаться средой, обсуждаются в разделе 12.2.

Требования

При традиционном подходе требования к системе разбиваются на требования к подсистемам, требования к подсистемам — на требования к компонентам, а требования к компонентам — на требования к блокам. Единый подход ко всем требованиям занимает многие рабочие часы, а затем время тратится на бумажную работу, которая связана с подробным сопоставлением и которая неминуемо пойдет насмарку позже, по мере изменения ведущих требований и понимания разработки.

В современном процессе требования к системе содержатся в общей концепции. Более низкие уровни требований управляются процессом — организованным в виде итераций, а не в виде компонентов более низких уровней — с помощью критериев оценки. Эти критерии обычно содержатся в наборе вариантов использования и в других представленных в текстовом виде целях. Общая концепция определяет соглашение между группой разработчиков и заказчиком. Эта информация должна изменяться, но изменяться медленно на протяжении всего жизненного цикла. Она представляется в виде, понятном заказчику. Критерии оценки содержатся в рабочих продуктах — спецификациях версий, которые отражают временные цели для данной итерации. Критерии оценки следуют как из общей концепции, так и из многих других источников, таких как

результаты анализа проблемы создания/покупки, вопросы управления рисками, архитектурные решения, ограничения при реализации, границы качества и даже смутные догадки.

Итерационные модели позволяют разработчику и заказчику работать с постоянно изменяющимися версиями системы. Практически требования могут и должны меняться вместе с изменением архитектуры и последовательными улучшениями приложения. В этом случае и заказчик, и разработчик обладают одинаковым, объективным пониманием приоритетов и соглашений по цене/срокам/производительности, связанных с этими требованиями. Вместо того чтобы сосредоточиваться на непротиворечивости, полноте и трассируемости спецификаций неустоявшихся требований, основное внимание в проектах должно уделяться получению правильных спецификаций общей концепции и преобразованию спецификаций более низких уровней посредством последовательного применения критериев оценки к итерациям, вносящим изменения в проект.

Этот подход в применении к поддержке управления требованиями имеет две составляющие:

1. Рекомендуемый подход к требованиям зависит как от представления в текстовом виде, так и от представлений, основанных на моделях. Следовательно, среда должна обеспечивать интегрированную автоматизацию документирования и визуального моделирования, чтобы позволить хранить спецификации в текстовом виде и в виде моделей вариантов использования. Необходимо управлять внесением и отслеживанием изменений в обеих формах и представлять их в виде — электронном или бумажном,— удобном для восприятия человеком.

2. Необходимо автоматизировать трассировку между требованиями и другими рабочими продуктами процесса. Степень трассируемости между различными комплектами является предметом давней дискуссии. Мое мнение таково, что комплект требований должен иметь четко определенную связь с рабочими продуктами тестирования, поскольку команда оценки несет ответственность за демонстрацию уровня соответствия продукта требованиям. Однако я не вижу никаких серьезных причин добиваться жесткого взаимного соответствия между рабочими продуктами комплекта требований и другими видами технических рабочих продуктов. Описание проблемной области, представленное в комплекте требований, и описание области решений, представленное в других комплектах технических рабочих продуктов, зачастую имеют трассировку, которую трудно представить. Прежде всего это справедливо для основанных на компонентах архитектур, в которых присутствует высокий процент коммерческих компонентов. Если процесс требует жесткой связи между требованиями и проектом, то архитектура, вероятнее всего, будет изменяться в сторону оптимизации трассировки требований, а не в сторону целостности проекта. Этот эффект становится еще более заметным при использовании инструментов, автоматизирующих процесс.

Проектирование

Инструменты, которые поддерживают процессы управления требованиями, проектирования, реализации и оценки, обычно используются совместно. Фактически, чем теснее они связаны, тем лучше. Основная поддержка, необходимая для проектирования,— это визуальное моделирование, которое применяется для создания моделей разработки, представления их в виде, пригодном для восприятия человеком, и для перевода в исходный код. Процесс с упреждающей разработкой архитектуры, основанный на демонстрациях, становится возможным благодаря наличию компонентов архитектуры и промежуточного ПО.

Реализация

Процесс реализации непосредственно основывается на среде программирования (редактор, компилятор, отладчик, загрузчик, поддержка во время выполнения). Но, кроме того, он должен быть интегрирован с инструментами, управляющими внесением изменений, с инструментами визуального моделирования и автоматизации тестирования.

Оценка и внедрение

Процесс оценки требует использования всех упомянутых выше инструментов, а также некоторых дополнительных — для автоматизации тестирования и для управления тестированием. Для упрощения внесения изменений проведение тестирования и изготовление документации должны быть автоматизированы. Отслеживание дефектов является еще одним важным инструментом, поддерживающим выполнение оценок: он обеспечивает инструментальное оформление, необходимое для автоматизации сбора параметров и контроля базовых версий. Этот инструмент требуется также для поддержки внедрения на протяжении всего жизненного цикла.

12.2 СРЕДА ПРОЕКТА

Среда проекта имеет три дискретных состояния. Это среда для создания прототипов, среда разработки и среда сопровождения.

1. *Среда для создания прототипов* включает в себя архитектурный «испытательный стенд» для тестирования, который используется для создания прототипа архитектуры проекта, являющегося основой для принятия технических решений в течение двух стадий жизненного цикла — начальной стадии и уточнения. Такая неформальная конфигурация инструментария должна поддерживать следующие виды деятельности:
   - Оценка производительности и анализ технических рисков
   - Оценка альтернатив создания/покупки и изучение применимости коммерческих продуктов

- Выбор решений, касающихся устойчивости к ошибкам и динамического изменения конфигурации
- Анализ рисков, связанных с переходом к полномасштабной реализации
- Разработка сценариев тестирования и инструментов анализа требований

2. *Среда разработки* должна включать в себя полный набор инструментов разработки, необходимых для поддержки различных рабочих процессов, а также для поддержки «круговой» разработки в максимально возможной степени.

3. *Среда сопровождения* обычно должна совпадать с окончательной версией среды разработки. В некоторых случаях среда сопровождения может являться подмножеством среды разработки, поставляемой в качестве одного из конечных продуктов проекта.

Переход к совершенному процессу разработки ПО создает новые проблемы и возможности для контроля за согласованностью действий и для оценки реального прогресса и качества. Опыт, полученный при осуществлении проектов в реальной жизни, показывает, что сильно интегрированная среда оказывается необходимой как для упрощения, так и для усиления контроля над процессом. С этой стороны существуют четыре важные дисциплины, которые критичны для управления и достижения успеха в современном процессе итерационной разработки:

1. Инструменты должны быть интегрированы для поддержки непротиворечивости и трассируемости. *«Круговая» разработка* — это термин, который используется для описания ключевого требования к различным видам среды, поддерживающим итерационную разработку.

2. *Управление изменениями* должно быть автоматизировано и построено таким образом, чтобы можно было управлять сразу несколькими итерациями и обеспечивалась простота внесения изменений. Изменение — это фундаментальная составляющая итерационной разработки.

3. Различные виды организационной *инфраструктуры* позволяют строить разные среды проекта на основе одних и тех же процессов и инструментов. Общая инфраструктура проектов способствует их согласованности, постоянному обучению персонала, использованию извлеченных уроков и получению иных стратегических выгод.

4. Распространение поддержки автоматизации на *среду других заинтересованных сторон* обеспечивает дальнейшее развитие безбумажного обмена информацией и более эффективное рассмотрение рабочих продуктов проекта.

12.2.1 «Круговая» разработка

По мере того как индустрия ПО движется в направлении автоматизированной поддержки различных рабочих продуктов, все более и более возрастает необходимость в гарантии эффективности и безошибочности переноса данных из одних рабочих продуктов в другие. «Круговая» разработка – это поддержка среды, необходимая для обеспечения непротиворечивости различных рабочих продуктов.

На рис. 12.2 изображены некоторые важные преобразования между репозиториями. Автоматизированная трансляция проектных моделей в исходный код (как при прямом, так и при обратном проектировании) определяется довольно четко. Автоматизированная трансляция проектных моделей в модели процесса (распределенной среды) также постепенно упрощается за счет использования таких технологий, как ActiveX и Common Object Request Broker Architecture (CORBA).

Компиляторы и загрузчики уже давно обеспечивают преобразование исходного кода в исполняемый код. По мере того как различные архитектуры начинают использовать разнородные компоненты, платформы и языки, сложность создания, управления и поддержания широкомасштабных сетей компонентов предъявляет новые требования к контролю за

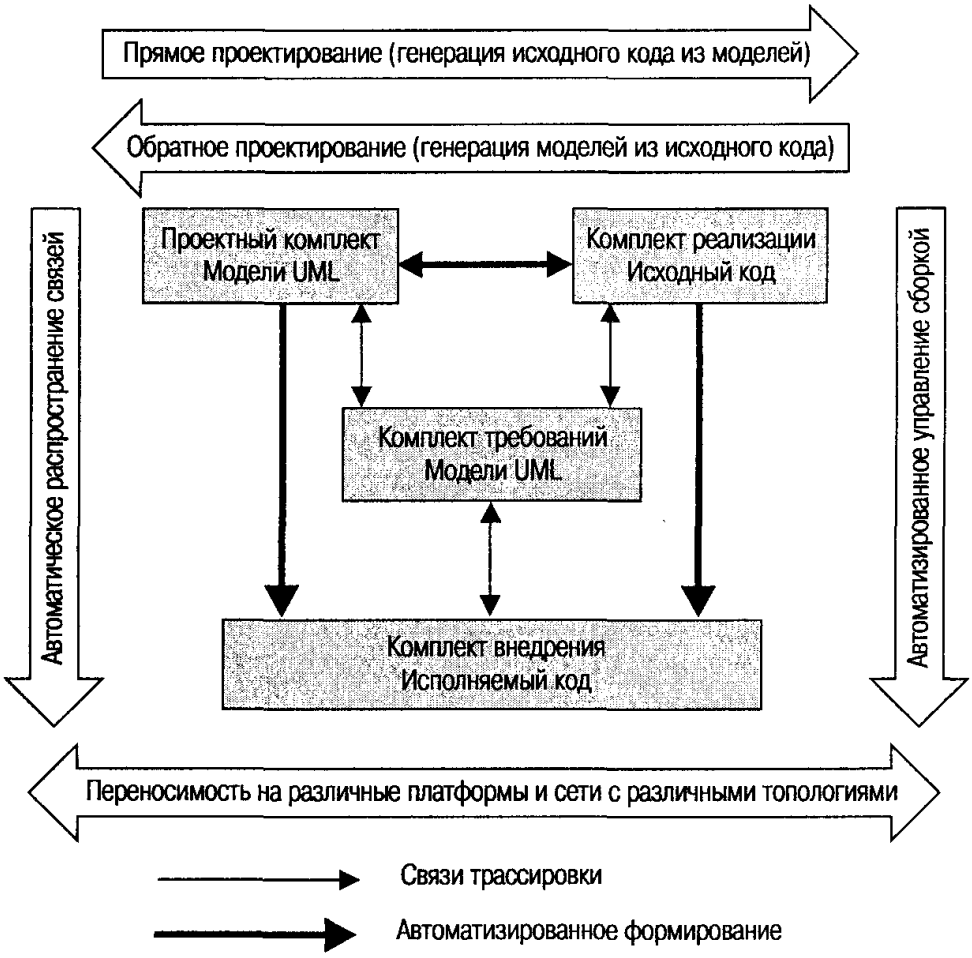


Рис. 12.2. *«Круговая» разработка*

конфигурацией и к автоматизации управления сборкой приложений из компонентов. Однако современные среды не поддерживают автоматизацию в максимально возможной степени. Например, автоматизированное создание вариантов тестирования из описаний варианта использования и сценария до сих пор не в состоянии поддерживать что-либо, кроме наиболее тривиальных случаев, таких как сценарии тестирования отдельных модулей.

Главная причина применения «круговой» разработки – это необходимость упростить внесение изменений в источники данных для разработки ПО. Такой контроль за конфигурацией всех технических рабочих продуктов важен для поддержания непротиворечивого и свободного от ошибок представления изменяющегося продукта. Однако нет никакой необходимости иметь переходы в двух направлениях во всех случаях. Например, хорошо было бы строить варианты тестирования для сценариев, описанных для заданного логического множества объектов, но мы не можем получать объекты исключительно на основании вариантов тестирования. Похожим образом, обратное превращение плохо написанного исходного кода в объектно-ориентированную проектную модель может оказаться непродуктивным.

Трансляция данных из одного источника в другой не всегда может быть выполнена на 100%. Например, перевод проектных моделей в исходный текст на C++ может касаться только структурного и декларативного аспектов представления исходного кода. Компоненты кода по-прежнему должны быть наполнены спецификой атрибутов или методов конкретного объекта.

12.2.2 Управление изменениями

Управление изменениями так же важно для итерационного процесса, как и планирование. Отслеживание изменений в технических рабочих продуктах является решающим для понимания реальных тенденций прогресса разработки и тенденций качества, направленных на получение приемлемого конечного продукта или промежуточной версии. В рамках традиционных процессов создания ПО способы управления конфигурацией технических рабочих продуктов использовались преимущественно на поздних этапах жизненного цикла. В современном процессе, где комплекты рабочих продуктов требований, проекта и реализации представляются посредством строгих нотаций уже на ранних стадиях жизненного цикла и претерпевают множество изменений, управление изменениями оказывается фундаментальным для всех стадий и практически для всех видов деятельности.

Запросы на внесение изменений

Наименьшая неделимая единица программистской работы, в результате которой производится создание, изменение или отказ от устаревших компонентов в рамках базовой конфигурации, называется запросом на внесение изменений в ПО (SCO, Software Change Order). Запросы на внесение изменений – один из главных механизмов распределения и

планирования работы по созданию ПО. Пример запроса, приведенный на рис. 12.3, является хорошей отправной точкой для описания набора примитивов, требуемых для внесения изменений. Здесь показан уровень детализации, необходимый для управления изменениями в современном процессе создания ПО. За счет автоматизации ввода данных и поддержки записей с изменениями в онлайновом режиме может быть автоматизирована и бюрократическая составляющая управления изменениями, связанная с различными отчетами о состоянии проекта.

Определение уровня, на котором следует писать SCO, всегда является проблемой. Что такое отдельное изменение? Является оно изменением модуля программы или компонента, файла или подсистемы? Является ли оно новой возможностью, исправлением дефекта или повышением производительности? Для большинства проектов неделимая единица измерения SCO принимается без особого труда. Вообще говоря, SCO должен быть написан для отдельного компонента, чтобы его выполнение можно было возложить на одного человека. Если исправления требуют труда двух человек из двух разных команд, необходимо писать два отдельных SCO.

Основными полями SCO являются название, описание, параметры, исправления, оценка и диспозиция.

- *Название.* Название предлагается автором и окончательно утверждается советом по контролю за конфигурацией (CCB, Configuration Control Board). В этом же поле должна содержаться ссылка на внешний отчет о проблеме, если это изменение было инициировано извне (например, пользователем).
- *Описание.* Описание проблемы включает в себя имя автора, дату возникновения, присвоенный CCB идентификатор и идентификаторы соответствующих версий вспомогательного ПО. Описание проблемы в текстовом виде должно содержать как можно больше деталей, к нему должны прилагаться участки программы, копии экрана, сообщения об ошибках и любые другие данные, которые могут помочь локализовать проблему или описать необходимые изменения.
- *Параметры.* Параметры, собираемые по каждому запросу, важны для планирования, составления графика и оценки качества. Категория вносимых изменений может иметь тип 0 (критичная ошибка), 1 (ошибка), 2 (усовершенствование), 3 (новая возможность) и 4 (другое). При принятии запроса выполняются предварительные оценки дефектов в ПО и объема работ, необходимого для решения проблемы. Графа «Дефект» является количественным выражением объема требуемых изменений, «Переделка» дает количественное определение сложности вносимых изменений. По мере внесения исправлений фиксируется реальный объем дефектного ПО и уточняются реальные объемы работ, необходимые для переделок. В графе «Анализ» указывается число человеко-часов, затраченных на понимание требуемых изменений (воспроизведение проблемы, ее локализация и устранение, если тип изменений 0 или 1; анализ и создание прототипов альтернативных решений, если тип равен 2 или 3). Графа «Реализация» определяет количество человеко-часов,

затраченных на проектирование и реализацию исправления. В графе «Тестирование» указывается количество часов, затраченных на тестирование исправлений, графа «Документация» определяет общий объем работ по обновлению других рабочих продуктов, таких как руководство пользователя или описание версии. Количество дефектного ПО задает глубину вносимых изменений и может выражаться в SLOC, функциональных точках, файлах, компонентах или классах. В случае выбора SLOC программа сравнения исходных файлов, которая выявляет количественные различия, может предоставить простую оценку объема дефектного ПО. Вообще говоря, точное значение числа дефектных строк не слишком важно. При изменении от 0 до 100 строк их количество следует округлять до ближайшего десятка, при изменении от 100 до 1000 строк — до ближайшей сотни и т.д.

- *Исправление.* Это поле включает в себя имя ответственного за реализацию вносимых изменений, изменяемые компоненты, реальные показатели и описание изменений. Уровень детализации компонентов, допускающий их передачу для внесения изменений, может варьироваться, но, вообще говоря, самый низкий уровень передаваемых компонентов должен быть приблизительно таким, чтобы работу мог выполнить один человек. Например, просто «компонент» является недостаточной детализацией для передачи команде.

- *Оценка.* Это поле описывает способ выполнения оценок: проверка, анализ, демонстрация или тестирование. В тех случаях, когда это возможно, следует ссылаться на все выполненные тесты как по уже существующим вариантам тестирования, так и по вновь созданным, и должны определяться все конфигурации, при которых выполнялось тестирование: платформы, топология и компиляторы.

- *Диспозиция.* ССВ присваивает запросу одно из следующих значений статуса:
  - Предложен: запрос написан и ожидает рассмотрения ССВ
  - Принят: ССВ утвердил запрос
  - Отклонен: запрос закрыт с указанием причин — например, не является проблемой, повтор, устаревшие изменения, выполнено по другому запросу
  - На хранение: запрос принят, но отложен до более поздней версии
  - В работе: запрос передан, и по нему ведется активная работа в организации-разработчике
  - На оценке: выполнен организацией-разработчиком; находится на оценке в организации, выполняющей тестирование
  - Закрыт: полностью выполнен, с чем согласны все члены ССВ

ССВ может также присвоить приоритет и идентификатор версии для определения приоритетов и организации параллельной разработки.

Название _____

Описание

Имя:_____ Дата: _____

Проект: _____

Параметры

Категория:_____

(0/1 – ошибка, 2 – усовершенствование, 3 – новая возможность, 4 – другое)

Предварительная оценка

Дефект:_____

Переделка: _____

Действительно необходимые доработки

Анализ:_____ Тестирование:_____

Реализация:_____ Документация:_____

Исправление

Аналитик:_____

Программный компонент:_____

Оценка

Метод:_____

(проверка, анализ, демонстрация, тестирование)

Тестирование выполнил:_____ Платформы:_____ Дата:_____

Диспозиция

Состояние:_____ Версия:_____ Приоритет:_____

Приемка:_____ Дата:_____

Закрытие:_____ Дата:_____

Рис. 12.3. *Элементарные составляющие, из которых состоит запрос на внесение изменений в ПО*

Базовая конфигурация

Базовая конфигурация — это поименованный набор программных компонентов вместе с необходимой документацией, который подвергается внесению изменений и обновляется, сопровождается, тестируется и признается морально устаревшим как единое целое. Для систем управления сложной конфигурацией существует множество стандартов, специфичных для данного проекта и для данной предметной области.

Существуют два класса базовых конфигураций: внешняя версия продукта и внутренние тестовые версии. Базовая конфигурация — это поименованный набор компонентов, рассматриваемый как единое целое. За этим следят формально, поскольку ее используют в качестве формы обмена между группами. Например, организация-разработчик может передать версию базовой конфигурации организации, выполняющей тестирование, и даже самой себе. При выполнении проекта версия базовой конфигурации может быть передана сообществу пользователей для бета-тестирования.

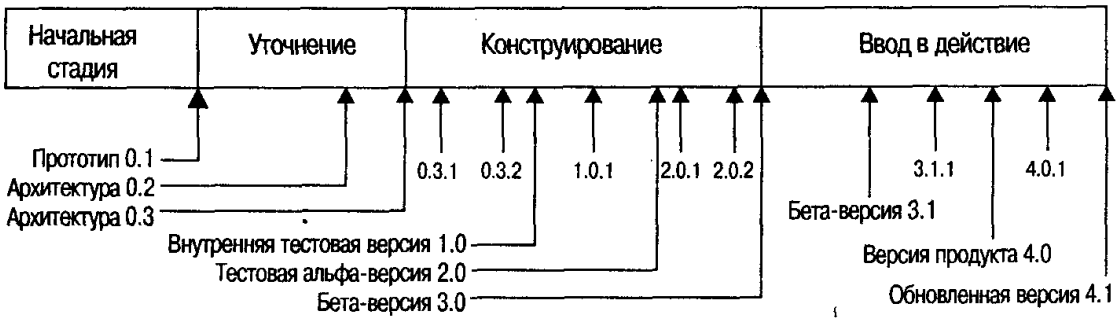
В общем случае для большинства систем требуются три уровня базовых версий: основные, второстепенные и промежуточные. Каждому уровню соответствует число в идентификаторе вида N.M.X, где N — номер основной версии, M — номер второстепенной версии, а X — идентификатор промежуточной версии. Основная версия представляет собой новое поколение продукта или проекта, в то время как второстепенная версия — это тот же основной продукт, но с улучшенными возможностями, производительностью или качеством. Основные и второстепенные версии обычно являются внешними версиями продукта, которые остаются неизменными и поддерживаются в течение некоторого периода времени. Промежуточная версия соответствует рабочей конфигурации, которую намереваются использовать как временную. Чем короче ее жизненный цикл, тем лучше. На рис. 12.4 показаны примеры историй некоторых версий для двух различных ситуаций.

Как только базовая версия ПО попадает под контроль, все изменения в нем начинают отслеживаться. Нужно различать причины, по которым возникает необходимость вносить изменения. Существуют следующие категории изменений:

- *Тип 0:* критичные ошибки — это такие ошибки, которые почти всегда фиксируются перед каждой внешней версией. Вообще говоря, этот вид изменений представляет собой фатальные ошибки, которые оказывают влияние на применимость ПО в наиболее важных вариантах использования.

- *Тип 1:* ошибка или дефект, который либо не ухудшает полезность системы, либо его можно обойти. Такие ошибки обычно связаны с некоторыми неудобствами в критичных вариантах использования или с серьезными дефектами во второстепенных вариантах использования, вероятность столкнуться с которыми сравнительно низка.

Типичная последовательность версий для единственного в своем роде крупномасштабного проекта



Типичная последовательность версий для небольшого коммерческого продукта

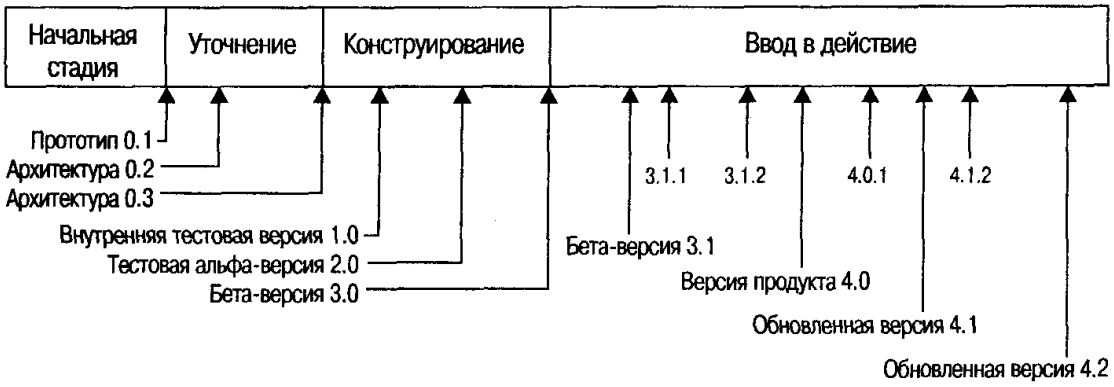


Рис. 12.4. *Примеры историй версий для типичного проекта и типичного продукта*

- *Тип 2:* изменение, которое можно рассматривать скорее как усовершенствование, нежели как реакцию на ошибку. Его целью, как правило, является повышение производительности, улучшение тестируемости, применимости или какого-либо другого аспекта качества, что является свидетельством высокого класса разработки.
- *Тип 3:* изменение, необходимость которого вызвана обновлением требований. Это могут быть новые свойства или функциональные возможности, находящиеся вне рамок текущей общей концепции и бизнес-плана.
- *Тип 4:* изменения, которые не подходят ни под одну из других категорий. Примеры подобных изменений включают в себя изменения только в документации или обновление версии под коммерческие компоненты.

В таблице 12.1 приведены примеры таких изменений в контексте двух различных проектных областей: крупномасштабная высоконадежная система контроля за воздушным движением и инструмент для разработки ПО в виде прикладного пакета.

Таблица 12.1.
Характерные примеры изменений для двух разных проектов

Тип изменений	Проект системы контроля за воздушным движением	Инструмент для визуального моделирования
Тип 0	Блокировка управления и потеря полетных данных	Потеря данных пользователя
Тип 1	Время ожидания ответа, превышающее указанное в требованиях на 0.5 с	Браузер разворачивает, но не сворачивает выведенные записи
Тип 2	Добавление внутреннего поля для вывода времени ожидания ответа	Использование различных цветов для того, чтобы можно было отличать изменения в визуальной модели, внесенные по сравнению с предыдущей версией
Тип 3	Увеличение числа полетов, которыми можно управлять одновременно, с 1200 до 2400	Переход на новую платформу, например WinNT
Тип 4	Переход с Oracle 7 на Oracle 8 для ускорения обработки запросов	Прерывание при обращении к MSExcel 5.0 из-за ошибки, возникающей при управлении ресурсами в Windows

Совет по контролю за конфигурацией

CCB (Configuration Control Board) – это группа людей, которая принимает ответственные решения относительно базовой конфигурации. В состав CCB обычно входят менеджер по созданию ПО, менеджер по созданию архитектуры ПО, менеджер по разработке ПО, менеджер по оценке ПО и другие заинтересованные стороны (заказчик, менеджер проекта по созданию ПО, системный инженер, пользователь), которые совместно отвечают за сопровождение поставляемой программной системы. Обычно CCB принимает меры в результате проведения совещаний. Но в определенных ситуациях могут иметь смысл распространение, согласование и утверждение действий CCB, выполняемые в онлайновом режиме.

В понятие итерационного процесса разработки должно входить всеобъемлющее и строгое управление изменениями в базовых версиях ПО. Фундаментальный процесс управления разработкой и сопровождения ПО описывается с помощью набора последовательных состояний, определяемых SCO. Слова в квадратных скобках ([]) описывают состояние SCO по мере его движения по процессу.

- [Предложен]. Предварительная версия предложенного изменения передана на рассмотрение CCB. В предлагаемое изменение должны входить техническое описание проблемы и приблизительная оценка необходимых для ее решения усилий.
- [Принят, на хранение, отклонен] CCB присваивает уникальный идентификатор каждому предложенному изменению и либо

принимает, либо сохраняет, либо отвергает это изменение. Принятие означает внесение изменений уже в следующей версии; принятие на хранение означает, что изменения приняты, но отложены до будущих версий; отклонение квалифицирует изменение как не заслуживающее внимания, лишнее с учетом других предложенных изменений или выходящее за рамки проекта. ССВ убеждается в том, что все поля запроса заполнены правильно и точно, перед тем как принять его, после чего запрос направляется лицу, ответственному за внесение изменений в организации-разработчике.

- [В работе]. Ответственное лицо выполняет анализ, реализацию и тестирование решения, удовлетворяющего запросу. Эта задача включает в себя обновление документации, описания версии и реальных значений параметров запроса, а также составление при необходимости новых запросов. Добившись полного решения, ответственное лицо заполняет раздел «Исправление» запроса и передает его для оценки независимой команде по тестированию.
- [На оценке]. Независимая команда по тестированию оценивает, полностью ли выполнен запрос. Если это так, запрос передается в ССВ для окончательного закрытия.
- [Закрыт]. Когда организация-разработчик, независимая тестирующая организация и ССВ соглашаются с тем, что запрос выполнен, он получает статус закрытого.

12.2.3 Различные виды инфраструктуры

С точки зрения автоматизации процесса организационная инфраструктура обеспечивает основные активы организации, в том числе два основных рабочих продукта: политику, включающую в себя стандарты процессов для проектов разработки ПО, и среду, содержащую перечень инструментов. Эти инструменты являются строительными «кирпичиками», из которых можно эффективно и экономично компоновать среду проекта.

Организационная политика

Организационная политика обычно представлена в виде справочного руководства, в котором дается определение элементарных понятий жизненного цикла и процесса (основные контрольные точки, промежуточные рабочие продукты, проектные репозитории, метрики, роли и ответственность). Справочное руководство содержит основу для ответа на следующие вопросы:

- Что делается? (виды деятельности и рабочие продукты)
- Когда это делается? (соответствие стадиям жизненного цикла и контрольным точкам)
- Кем это делается? (распределение ролей и ответственности в команде)

- Как убедиться в адекватности? (контрольные точки, параметры и стандарты)

Требование баланса является важным соображением при определении организационной политики. Нередко организации впадают в одну из двух крайностей: отсутствие установленного процесса или чересчур большое количество стандартизации и бюрократии. Эффективные виды организационной политики имеют несколько общих особенностей:

- Они кратки и избегают положений о политике, для описания которых требуются документы пятнадцатисантиметровой толщины.
- Они облекают политику в реальные *обязательства*, а затем способствуют их выполнению.
- Они избегают использования в положениях о политике слова «следовало бы». Вместо меню возможностей («следовало бы») политика нуждается в кратком перечне обязательных стандартов («следует»).
- Отказ от отдельных положений является не правилом, а исключением.
- Соответствующая политика пишется на соответствующем уровне.

Последний пункт заслуживает более подробного обсуждения. В индустрии, связанной с разработкой ПО, существует множество разных организационных структур. Компании, интенсивно разрабатывающие ПО, обладают тремя различными уровнями организации, на каждом из которых политика уделяет основное внимание определенным аспектам:

- Высший организационный уровень: стандарты, которые способствуют (1) стратегическим и долговременным улучшениям процесса, (2) использованию общей технологии и обучения, (3) сравнимости работы подразделения, осуществляющего проект, с работой подразделения, занимающегося бизнесом, и (4) обязательному контролю качества.
- Средний уровень основной деятельности организации: стандарты, которые способствуют (1) тактическим и кратковременным улучшениям процесса, (2) внедрению и обучению специфической для данной области технологии, (3) повторному использованию компонентов, процессов, обучения, инструментов и опыта сотрудников и (4) соответствию стандартам, принятым в организации.
- Низший уровень проекта: стандарты, которые способствуют (1) достижению поставленных целей по качеству, срокам и затратам, (2) специальному обучению для данного проекта, (3) соответствию требованиям заказчика и (4) соответствию стандартам, принятым в организации/подразделении.

В общем случае стандартизация должна сосредоточиваться на подразделениях основной деятельности организации, а не на высшем уровне или уровне проектов. Эффект от стандартизации наиболее заметен именно на том уровне, где присутствует наибольшая общность и возможность

повторного использования в проектах, процессах и инструментах. Стандартизация способов и инструментов для разработки ПО в рамках различных областей деятельности – дело весьма сложное, поскольку приоритеты процесса, инструменты, способы, методы и культура заинтересованных сторон могут быть совершенно различными. Попытки стандартизации внутри нескольких областей, имеющих между собой мало общего, приводят либо к совершенно размытым положениям политики, либо к отказу от отдельных положений, которые используются чересчур часто. Стандартизация на слишком высоком уровне также весьма проблематична. Если все команды разработчиков оставить один на один со своими задачами, то все процессы и среды проекта вскоре окажутся локально оптимизированными. Со временем вся инфраструктура организации по улучшению и усовершенствованию процесса будет выхолощена.

Организационная политика является определяющим документом для всех вариантов политики организации в области создания ПО. При любой оценке процесса она представляет собой основное руководство к действию. Используя этот документ, каждый получает возможность исследовать и изучать проекты и персонал с целью выяснения, выполняет ли организация то, что она декларирует. На рис. 12.5 показаны общие контуры организационной политики.

I. Определения элементарных составляющих процесса

- A. Стадии жизненного цикла (начальная стадия, уточнение, конструирование, ввод в действие)
- B. Контрольные точки (основные и второстепенные, оценки состояния)
- C. Рабочие продукты (комплекты требований, проекта, реализации, внедрения, управления)

II. Различные виды организационной политики в области создания ПО

- A. Декомпозиция работ
- B. План разработки ПО
- C. Управление изменениями
- D. Метрики ПО
- E. Среда разработки
- F. Критерии оценки и критерии приемки
- G. Управление рисками
- H. Тестирование и оценка

III. Политика отказа

IV. Приложения

- A. Оценка текущего процесса
- B. План усовершенствования процесса создания ПО

Рис. 12.5. *Контуры организационной политики*

Организационная среда

Организационная среда для автоматизации стандартного процесса дает ответы на многие вопросы относительно того, как практически устроены различные вещи, а также предоставляет инструменты и способы для автоматизации процесса. Ниже приводятся типичные компоненты строительных «кирпичиков» для автоматизации деятельности:

- Стандартизированный выбор инструментов (посредством инвестирования конкретной лицензии либо посредством переговоров с поставщиками о предоставлении выгодных скидок с тем, чтобы у команд, работающих над проектом, появлялась экономическая мотивация использовать именно этот инструмент), который способствует развитию основных рабочих процессов и повышает ROI при обучении.
- Стандартные нотации для рабочих продуктов, такие как UML для всех проектных моделей или Ada 95 для всех разрабатываемых на заказ критичных по надежности рабочих продуктов.
- Дополнения к инструментам, например заранее разработанные или выполненные на заказ шаблоны (описание архитектуры, критерий оценки, описания версий, оценки состояния).
- Шаблоны для различных видов деятельности (планирование итерации, деятельность в рамках основной контрольной точки, деятельность советов по контролю за конфигурацией).
- Другие компоненты организационной инфраструктуры, польза от которых является косвенной:
  - Библиотека ссылок на предшествующий опыт планирования, оценки и улучшения параметров процесса; ответы на вопросы «Насколько хорошо?», «Как много?», «Почему?»
  - Существующие практические примеры, включая объективные точки отсчета для оценки успешно завершившихся проектов, которые следовали принципам организационного процесса
  - Библиотека примеров рабочих продуктов проекта, таких как планы разработки ПО, описания архитектуры и истории оценок состояния
  - Модели проведения аудита и способы сопоставления с внешними схемами оценки, такими как Software Engineering Institute's Capability Maturity Model (SEI CMM).

12.2.4 Среда для других заинтересованных сторон

При переходе к современному итерационному процессу разработки, поддерживаемому автоматизацией, не следует ограничиваться только командой разработчиков. Многие широкомасштабные проекты, выполняемые по контрактам, вовлекают людей из сторонних организаций, которые представляют другие заинтересованные стороны, принимающие участие

в процессе разработки. Среди них могут быть наблюдатели за ходом выполнения контракта из посреднической организации, персонал конечного пользователя по инженерному сопровождению продукта, сторонние подрядчики по сопровождению продукта, независимые подрядчики, выполняющие верификацию и аттестацию, представители регулирующего органа и др.

Представителям других заинтересованных сторон тоже необходим доступ к ресурсам среды разработки для того, чтобы они могли вносить свой вклад в общее дело. Если у внешней команды, представляющей ту или иную заинтересованную сторону, отсутствуют ресурсы среды, позволяющие им принимать рабочие продукты в онлайновом режиме, единственным средством для обмена информацией является бумага. Такая ситуация приводит к проблемам, которые описаны в главе 6 как присущие традиционному процессу.

Среда, допускающая доступ заинтересованных сторон извне в онлайновом режиме, позволяет им принимать участие в процессе следующим образом:

- Получать и использовать работающие продукты для выполнения оценок вручную
- Использовать в онлайновом режиме те же инструменты, данные и отчеты, которые организация-разработчик применяет для управления и наблюдения за проектом
- Исключить излишние поездки, задержки, связанные с обменом бумагами и переводом в другие форматы, расходы на бумагу и доставку, а также другие накладные расходы

На рис. 12.6 показаны новые возможности для производительной деятельности третьих заинтересованных сторон при выполнении больших контрактов. Существует несколько важных причин для распространения ресурсов среды разработки на некоторые заинтересованные стороны.

- Технические рабочие продукты — это не только бумага. Материалы в электронном виде в строгой нотации, например визуальные модели или исходный код, могут рассматриваться более эффективно при использовании инструментов вместе с подходящими браузерами.
- Независимые оценки изменяющихся рабочих продуктов становятся возможными благодаря электронному доступу в режиме чтения к текущим данным, таким как библиотеки базовой конфигурации и база данных управления изменениями. Рассмотрения и проверки, оценки дефектного кода и переделок, анализ параметров и даже бета-тестирование могут выполняться независимо от команды разработчиков.
- Даже бумажные документы должны доставляться электронным способом для снижения стоимости продукции и времени оборота.

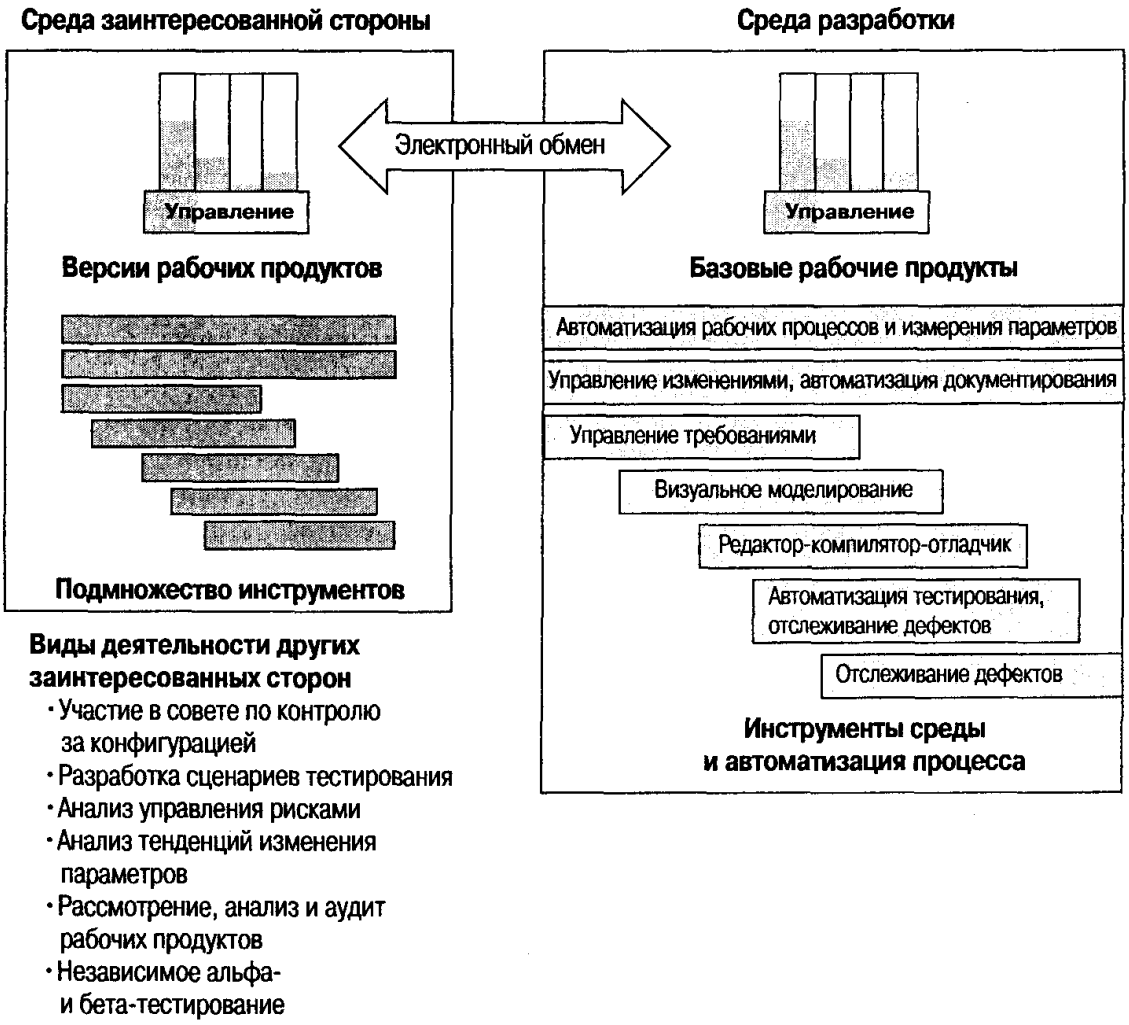


Рис. 12.6. *Распространение среды на другие заинтересованные стороны*

Как только заинтересованные стороны получают электронный доступ к ресурсам среды, постоянная и надежная обратная связь становится более эффективной, осязаемой и полезной. Команда разработчиков должна создавать открытую среду и предоставлять адекватные ресурсы, обеспечивающие доступ заказчику. Сами заинтересованные стороны не должны злоупотреблять таким доступом, обязаны принимать плодотворное участие и избегать прерывания разработки. Интернет- и интранет-технологии делают безбумажную среду экономически выгодной.

Распространение ресурсов среды на другие заинтересованные стороны создает ряд проблем. Какую степень свободы доступа следует поддерживать? Кто будет финансировать вложения в инструменты и среду? Насколько безопасен обмен информацией? Как синхронизировать внесение изменений? Некоторые из этих проблем обсуждаются в главе 17.

Глава 13

Метрики и контроль качества

Главной темой современного процесса разработки ПО является решение основных проблем управления сложным ПО:

1. Правильное выполнение разработки, при котором внимание уделяется прежде всего архитектуре
2. Управление рисками в рамках итерационного процесса
3. Уменьшение сложности с помощью технологий, основанных на компонентах

Ключевые моменты

▲ Прогресс в достижении целей проекта и качество программных продуктов должно подвергаться измерению на протяжении всего цикла разработки ПО.

▲ Значения параметров чрезвычайно важны для управления процессом. Не менее важны тенденции их изменения.

▲ Наиболее ценные параметры извлекаются непосредственно из рабочих продуктов.

▲ Объективный анализ и автоматизированный сбор информации важны для успешной работы с любыми метриками. Субъективные оценки и сбор информации вручную приведут, скорее всего, к неудаче.

4. Создание ситуации, при которой прогресс в разработке ПО и повышение качества становятся осязаемыми за счет автоматизированного управления изменениями.
5. Автоматизация вспомогательной деятельности посредством «круговой» разработки и интегрированной среды

Четвертый пункт этого списка является темой настоящей главы. Принципиально трудно управлять тем, что не может быть объективно измерено. Это было одной из основных проблем традиционного процесса создания ПО, в котором промежуточные продукты формировались преимущественно в виде бумажных документов. Параметры ПО (метрики)

являются инструментом для осуществления деятельности и получения рабочих продуктов в процессе разработки/интеграции ПО. Успех любого процесса по созданию ПО, в котором контроль за параметрами выполняется вручную и требует больших трудозатрат, будет ограниченным. В современном процессе разработки наиболее важными параметрами ПО являются простые и объективные измерения того, как продукт и проект изменяются с различных точек зрения.

Необходимо иметь возможность измерять качество программных продуктов и прогресс на пути к достижению целей проекта на протяжении всего цикла разработки ПО. Целью метрик ПО является обеспечение команды разработчиков и управляющей команды:

- Точной текущей оценкой прогресса
- Уровнем качества изменяющегося программного продукта
- Основой для приблизительной оценки затрат и времени, необходимых для получения конечного продукта, с учетом того, что точность оценки увеличивается со временем

13.1 СЕМЬ ОСНОВНЫХ МЕТРИК

Существует множество различных метрик, которые могут представлять ценность для управления современным процессом. Я определяю семь основных метрик, которые следует использовать во всех проектах по созданию ПО. Три из них являются показателями управления, а четыре — показателями качества.

Показатели управления

- Работа и прогресс (работа, выполненная к настоящему моменту)
- Предусмотренные в бюджете расходы и затраты (расходы, понесенные к настоящему моменту)
- Динамика изменений в проектных командах и в штатном расписании (изменения в составе персонала, произошедшие к настоящему моменту)

Показатели качества

- Интенсивность изменений и стабильность (интенсивность изменений к настоящему моменту)
- Дефекты и коэффициент дефектности (среднее количество дефектов на одно изменение к настоящему моменту)
- Доработки и адаптируемость (средний объем доработок на одно изменение к настоящему моменту)
- Среднее время наработки на отказ (Mean time between failures, MTBF) и завершенность (уровень дефектов к настоящему моменту)

В таблице 13.1 описаны эти основные метрики. Каждая из них имеет два измерения: статическое *значение*, используемое в качестве цели, и динамическая *тенденция*, которая применяется для достижения цели. Значения метрик дают представление только в одном измерении; с точки зрения управления процессом более важными оказываются тенденции их изменения. Тенденции изменения метрик во времени позволяют понять, в каких направлениях изменяются продукт и процесс. Итерационному процессу присуще внесение изменений, а измерение этих изменений является важным аспектом параметров программы. Абсолютные значения производительности и качественных улучшений вторичны до тех пор, пока не будет решена фундаментальная задача управления: предсказуемость затрат средств и времени для заданного уровня качества.

Таблица 13.1.
Семь основных метрик

Метрика	Цель	Способы определения
Работа и прогресс	Планирование итераций, сравнение плана и реальных достижений, показатель управления	SLOC, функциональные точки, объектные точки, сценарии, варианты тестирования, SCO
Предусмотренные в бюджете расходы и затраты	Понимание финансовых вопросов, сравнение плана и реальных значений, показатель управления	Ежемесячные затраты, число используемых в течение месяца сотрудников с полным рабочим днем, процент израсходованного бюджета
Динамика команд и штатного расписания	План потребления ресурсов по сравнению с реальностью, уровень найма, уровень сокращения численности персонала	Число новых сотрудников за месяц, число ушедших сотрудников за месяц
Интенсивность изменений и стабильность	Планирование итераций, показатель соблюдения графика	Количество поданных SCO по сравнению с количеством закрытых SCO по типам (0,1,2,3,4), приходящихся на одну версию/компонент/подсистему
Дефекты и коэффициент дефектности	Достижение цели, доработка ПО, показатель качества	Количество переделанных SLOC на одно изменение по типам (0,1,2,3,4), приходящихся на одну версию/компонент/подсистему
Доработки и адаптируемость	Достижение цели, переделка ПО, показатель качества	Среднее количество часов, затраченных на одно изменение по типам (0,1,2,3,4), приходящихся на одну версию/компонент/подсистему
MTBF и завершенность	Покрытие/адекватность тестирования, простота использования, показатель качества	Число отказов, количество часов между отказами при тестировании, приходящихся на одну версию/компонент/подсистему

В приложении C даются краткое обоснование и подробное описание этих метрик. Они были проверены на практике для проектов, в которых использовался итерационный подход. Практический пример в приложении D представляет собой подробное описание того, как эти метрики работают в реальном проекте.

Семь основных метрик могут быть использованы бесконечным числом способов для управления проектами и организациями. В процессе разработки проекта данные по предыдущим итерациям и проектам являются предварительными данными для планирования последующих итераций и проектов. Соответственно, после того как семейство метрик определено, они могут быть использованы проектом или организацией для предсказания затрат, сроков и качества выполнения различных работ в будущем.

Семь основных метрик основываются на здравом смысле и практическом опыте, связанном как с успехами, так и с неудачами. Они обладают следующими свойствами:

- Просты, объективны; их легко собирать, легко интерпретировать и трудно интерпретировать неправильно.
- Их сбор может быть автоматизирован, и он не нарушает нормальную работу системы.
- Они позволяют получать непротиворечивые оценки на протяжении всего жизненного цикла и выводятся непосредственно из самого состояния продукта, а не из субъективных оценок.
- Они полезны как для управленческого, так и для инженерного персонала, являясь средством обмена информацией о ходе работ и качестве, представленной в согласованном формате.
- Их точность растет на протяжении жизненного цикла.

Последнее свойство важно и достойно более подробного обсуждения. Метрики, используемые на стадии разработки (на которой преобладают интеллектуальная свобода и разрешение рисков), оказываются менее точными, чем те, что применяются на стадии производства (с преобладанием работ по реализации и управлению изменениями). Следовательно, заранее определенные метрики адаптируются к стадии производства, где высока рискованность затрат и возрастает значение управления. Задание метрик на стадии разработки направлено в основном на определение первоначальной базовой архитектуры и ожиданий, связанных с планом стадии производства.

13.2 ПОКАЗАТЕЛИ УПРАВЛЕНИЯ

Применяются три фундаментальных набора метрик управления: технический прогресс, финансовое состояние и прогресс в найме персонала. Изучая эти аспекты, управляющие проектом могут оценить, укладывается ли проект в бюджет и в сроки. Финансовое состояние – это то, что хорошо понятно всем; оно существовало всегда. Большинство менеджеров

имеют информацию о расходовании ресурсов в терминах финансовых затрат и сроков. Проблема заключается в том, чтобы оценить величину технического прогресса. В традиционных проектах, промежуточными продуктами которых являлись исключительно бумаги, полагались на субъективные оценки или на количество завершенных документов. Эти материалы отражали затраты, но были не слишком показательны относительно полезности выполненной работы.

Показатели управления, рекомендуемые в настоящей работе, включают в себя стандартное финансовое состояние, которое основано на системе приобретенной стоимости, объективных параметрах технического прогресса, адаптированных к основным критериям измерения для каждой большой команды или организации, и параметры, касающиеся персонала, которые позволяют понять динамику внутри команд.

13.2.1 Работа и прогресс

Различные виды деятельности при итерационной разработке проекта могут быть измерены посредством определения объема планируемой работы, выраженного в объективных единицах измерения, с дальнейшим отслеживанием прогресса (объем работы, выполненной к настоящему моменту) по отношению к плану (см. рис. 13.1). У каждой большой команды в организации должен существовать по крайней мере один аспект прогресса, относительно которого проводятся измерения. Для обычных команд (см. главу 11) стандартными аспектами могли бы быть:

- Команда, занимающаяся архитектурой ПО: продемонстрированные варианты использования
- Команда, занимающаяся разработкой ПО: количество SLOC, подпадающих под управление изменениями, количество закрытых SCO
- Команда, занимающаяся оценкой ПО: число открытых SCO, время (в часах) выполненного тестирования, соответствие критериям оценки
- Команда, занимающаяся управлением созданием ПО: число пройденных контрольных точек

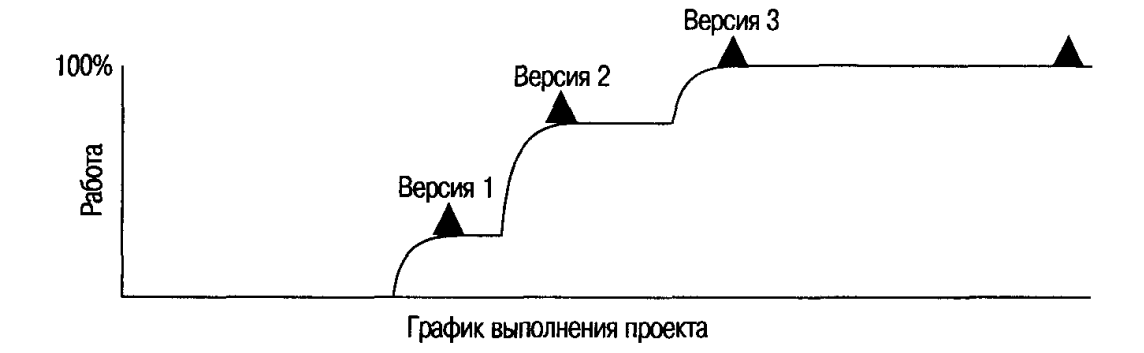


Рис. 13.1. *Ожидаемый прогресс типичного проекта с тремя основными версиями*

13.2.2 Предусмотренные в бюджете расходы и затраты

Для контроля над проектом требуется постоянно вычислять затраты на протяжении всего жизненного цикла. Использование метрик работы и прогресса может дать более объективную оценку технического прогресса для ее сравнения с финансовыми затратами. Для итерационного процесса разработки представляется важным детально планировать ближайшие действия (обычно для временного интервала менее шести месяцев) и составлять планы дальнейших действий в виде грубых оценок, которые должны уточняться по мере того, как истекает интервал времени текущей итерации и становится критичным планирование следующей итерации.

Отслеживание финансового прогресса обычно принимает форму, характерную для данной организации. Единственным общим подходом к измерению финансового состояния является применение системы приобретенной стоимости, которая позволяет детально определить затраты времени. Его недостатком при использовании для проектов по созданию ПО традиционно является невозможность объективной и точной оценки технического прогресса (процента завершенной работы). Это свойственно стадии разработки проекта. Тем не менее системы приобретенной стоимости доказали свою эффективность на стадии производства, для которой характерны правильное отслеживание реальных показателей по сравнению с планами и предсказуемые результаты. Другие метрики обеспечивают основу для получения подробных и реалистичных количественных данных по обратной связи для планирования и сверки, особенно на стадии производства, где финансовые и временные затраты являются наивысшими.

Современные процессы создания ПО легко поддаются измерению финансового состояния посредством подхода приобретенной стоимости. Основными параметрами системы приобретенной стоимости, обычно исчисляемой в долларах, являются следующие:

- План расходов: вид кривой планируемых для проекта затрат относительно планируемых сроков. Для большинства проектов по созданию ПО (и других трудоемких проектов) вид этой кривой повторяет вид кривой найма персонала.

- Реальный прогресс: техническое выполнение относительно планируемого прогресса, который лежит в основе кривой финансовых затрат. В нормально развивающемся проекте реальный прогресс неотступно следует за планируемым прогрессом.

- Реальная стоимость: кривая реальных затрат на проект для реального графика выполнения. В нормально развивающемся проекте эта кривая неотступно следует за планируемой кривой.

- Приобретенная стоимость: определяется стоимостью реального прогресса.

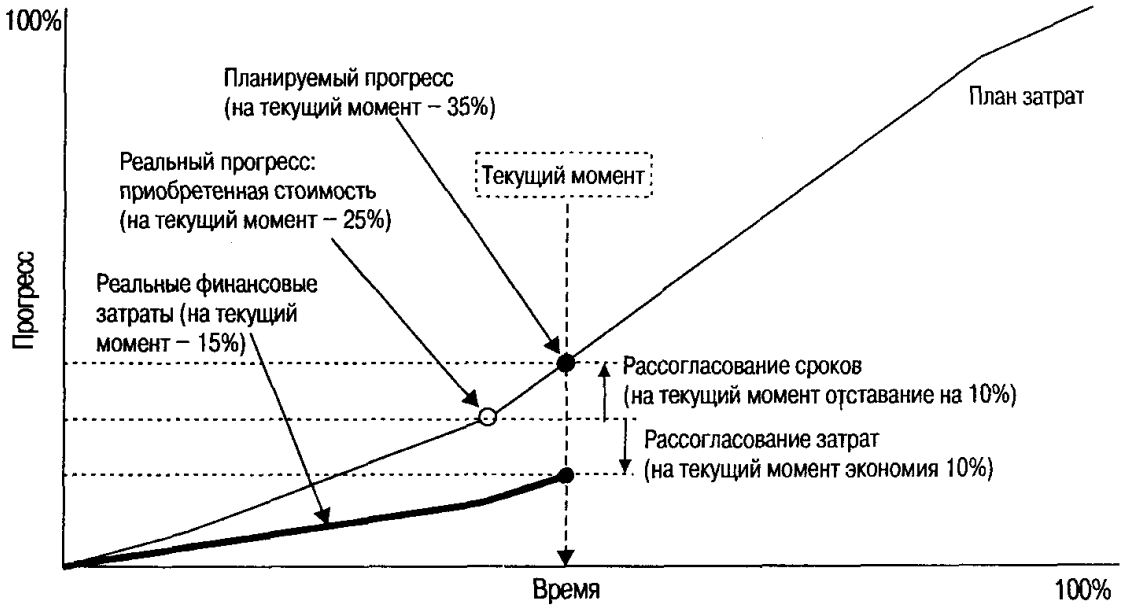


Рис. 13.2. *Основные параметры системы приобретенной стоимости*

- Рассогласование затрат: разность между реальными затратами и приобретенной стоимостью. Положительные значения относятся к ситуациям, связанным с перерасходом бюджета; отрицательные значения соответствуют ситуациям, связанным с экономией бюджета.
- Рассогласование сроков: разница между планируемыми затратами и приобретенной стоимостью. Положительные значения относятся к ситуациям, связанным с отставанием от графика; отрицательные значения соответствуют ситуациям, связанным с опережением графика.

На рис. 13.2 дается графическое представление этих параметров и приводится простой пример состояния проекта.

Основной целью других основных метрик является обеспечение команд по управлению и разработке более объективным подходом к оценке реального прогресса с наибольшей точностью. Из всех параметров системы приобретенной стоимости реальный прогресс является наиболее субъективной оценкой. Поскольку большинство менеджеров точно знают, сколько затрат они произвели и какая часть времени использована, то разногласия при выполнении точных оценок финансового благополучия касаются правильности оценки реального прогресса.

Для пояснения сильных и слабых сторон системы приобретенной стоимости рассмотрим процесс разработки этой книги, который во многих отношениях напоминает процесс разработки ПО. Реальный прогресс может быть оценен по текущему состоянию каждой главы, которое определяется относительно количества страниц, запланированных для данной главы. Я отслеживал состояние каждой части (являющейся связной последовательностью глав), используя следующие определения состояний и приобретенную стоимость (приобретенный процент завершенности):

- от 0 до 50%: содержимое не готово
- 50%: содержимое в черновом виде; автор завершил первую черновую версию текста и рисунков
- 65%: начальная версия текста; завершено предварительное редактирование текста
- 75%: версия, подлежащая рецензированию; завершено редактирование текста и рисунков
- 80%: обновленная версия; проверена взаимная согласованность глав
- 90%: рецензированная версия; автор включил в текст комментарии внешних рецензентов
- 100%: окончательная редакция; редактор завершил чистовую обработку текста

Оценки «процента завершенности» выполнялись субъективно на базе моего опыта написания сложных документов. Я планировал работать в течение 10 месяцев и потратить $10 000 на вспомогательные мероприятия. В таблице 13.2 приведены мой прогресс и соответствующая приобретенная стоимость на четвертом месяце работы. Хотя уже было написано вчерне около 400 страниц из планировавшихся 425, я оценивал завершенность своего труда только в 60%, используя средневзвешенное значение.

Таблица 13.2
Определение реального прогресса при разработке книги (пример)

Компонент	Страницы	%	Состояние
Часть I	60	75%	Версия, подлежащая рецензированию
Часть II	75	75%	Версия, подлежащая рецензированию
Часть III	90	65%	Начальная версия текста
Часть IV	30	65%	Начальная версия текста
Часть V	130	50%	Черновик
Остальное (предисловие, глоссарий, индекс)	40	25%	Содержимое не готово

Если построить кривую зависимости планируемых затрат от времени, подобную изображенной на рис. 13.2, то можно будет оценить, нахожусь ли я в рамках бюджета и в рамках графика. На рис. 13.3 показаны мой план и оценка на четвертом месяце, когда я опережал свой график на 20% и потратил на 30% меньше запланированного бюджета.

Этот пример является хорошей отправной точкой для обсуждения ключевых атрибутов планирования и оценки реального прогресса: создания объективной основы, описания подходящей декомпозиции работ и планирования с надлежащим качеством.

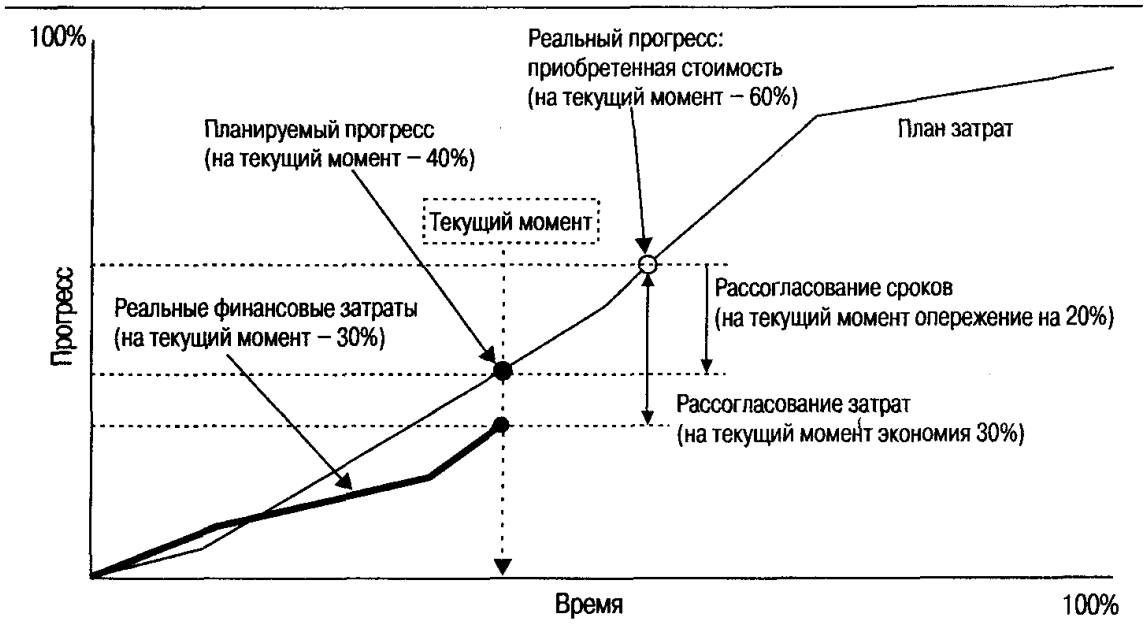


Рис. 13.3. *Оценка прогресса при написании книги (пример)*

Я ввел объективные критерии для определения процента завершенности данного компонента. (Для другого автора такие критерии могут оказаться неоптимальными.) Они основаны на моем опыте, на моем собственном стиле разработки и на абсолютно понятных взаимоотношениях с техническим редактором. Аналогично, для проектов по созданию ПО культура команды, опыт команды и стиль разработки (процесс, его строгость и его зрелость) будут управлять выбором критериев, используемых для объективной оценки прогресса.

Я выполнил декомпозицию работы, разбив ее на части (группы глав), что является самым простым подходом к отслеживанию прогресса. Так же, как и для ПО, это было совершенно естественным при наличии хорошо разработанной архитектуры книги. Тем не менее мне пришлось несколько раз изменять архитектуру (схему и направление) в первые месяцы. Детальная компонентная проработка на ранних стадиях жизненного цикла написания книги привела бы к необходимости малопривлекательных переделок и отвлекла бы меня от улучшения архитектуры. Более пригодная для отслеживания прогресса проекта в целом декомпозиция работ (учитывающая вклад автора, редактора, художников, рецензентов, составителей и издателя) должна организовываться самим процессом, при этом прогресс в создании должен явно отслеживаться только для отдельных компонентов.

Я планировал работу с точностью, подходящей для проекта, выполняемого одним лицом. Я принял решение не заниматься подробным отслеживанием прогресса, пока не будет полностью написана черновая версия компонента. На ранних стадиях я отслеживал прогресс с помощью простой субъективной оценки степени завершенности начальных черновых материалов. В том же духе следует работать и с большими проектами, используя такой уровень точности планирования, который согласуется с текущим состоянием проекта и с вероятностью повторного планирования.

13.2.3 Динамика изменений в командах и штатном расписании

Итерационная разработка должна сначала вестись небольшой командой до тех пор, пока все риски в требованиях и архитектуре не будут разрешены подходящим образом. В зависимости от наложения итераций и других характерных для данного проекта обстоятельств штатное расписание может меняться. Типичное для выполнения отдельных одноразовых задач по разработке (например, для построения корпоративной информационной системы) изменение численности задействованных в проекте сотрудников приведено на рис. 13.4. Разумно ожидать, что группа сопровождения для такого рода проектов окажется меньше группы разработки. При создании коммерческого продукта размеры групп поддержки и разработки могут оказаться одинаковыми. Когда долгоживущие, постоянно улучшаемые продукты подвергаются изменениям, сопровождение является всего лишь постоянным созданием новых улучшенных версий.

Отслеживание реального найма сотрудников по сравнению с планируемым является необходимым и хорошо понимаемым параметром управления. Существует еще один важный для управления показатель изменения движущей силы проекта — отношение увольняющихся и вновь принимаемых. Увеличение штата может замедлить общий прогресс проекта, поскольку давно работающим сотрудникам придется тратить производительное время на введение новых сотрудников в курс дела. Низкий процент ухода хороших сотрудников является признаком успеха. Сильной мотивацией для разработчиков является возможность привести что-либо в действие; это постоянная тема, лежащая в основе эффективного итерационного процесса разработки. Если такая мотивация отсутствует, хорошие разработчики уходят в другое место. Рост незапланированного увольнения людей — т.е. числа людей, преждевременно покидающих проект, — является одним из наиболее наглядных показателей того, что

Рис. 13.4. *Типичное изменение численности сотрудников*

этот проект ожидают проблемы. Причины преждевременного увольнения могут быть различными, но обычно это неудовлетворенность персонала методами управления, отсутствие командной работы или высокая вероятность провала в процессе достижения поставленных целей.

13.3 ПОКАЗАТЕЛИ КАЧЕСТВА

Четыре показателя качества основаны прежде всего на измерении изменений в ПО в процессе изменения базовых данных, имеющих отношение к разработке (таких, как проектные модели и исходный код). Эти метрики более полно рассматриваются в приложении С.

13.3.1 Интенсивность изменений и стабильность

Общая интенсивность изменений – это особый показатель прогресса и качества. Интенсивность изменений вычисляется как число запросов на внесение изменений в ПО, открытых и закрытых на протяжении всего жизненного цикла (см. рис. 13.5). Этот параметр может определяться в зависимости от типа вносимых изменений в расчете на одну версию, на все версии, на одну команду, на один компонент, на одну подсистему и т.д. Рассматриваемый совместно с метриками работы и прогресса, он позволяет оценить стабильность ПО и его движение в сторону стабильности (или нестабильности). Стабильность определяется как отношение между открытыми и закрытыми SCO. Интенсивность изменений, отнесенная к графику версии, дает представление о предсказуемости графика работ, что является основной ценностью данной метрики и показателем того, насколько хорошо идет процесс. Следующие три метрики качества имеют большее отношение к качеству продукта.

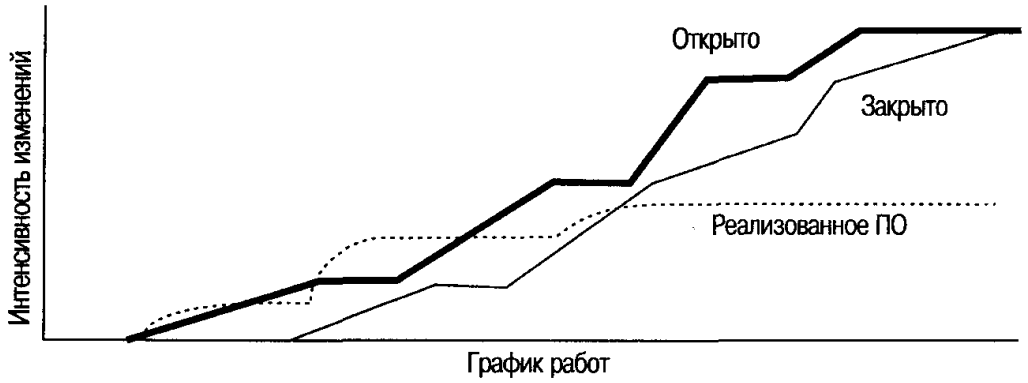


Рис. 13.5. *Ожидаемая стабильность на протяжении жизненного цикла благополучного проекта*

13.3.2 Дефекты и коэффициент дефектности

Дефектность определяется как средняя мера изменений, которая представляет собой объем базового ПО, требующего доработки (может выражаться в SLOC, функциональных точках, компонентах, подсистемах, файлах и т.д.). *Коэффициент дефектности* – это тенденция изменения

среднего количества дефектов с течением времени. Для благополучного проекта ожидаемая тенденция — уменьшение или стабильность (см. рис. 13.6).

Этот показатель позволяет понять, доброкачественный или злокачественный характер носят изменения ПО. В случае зрелого процесса итерационной разработки изменения на ранних этапах обычно приводят к большему объему отбраковки, чем на более поздних. Тенденция к увеличению дефектов с течением времени ясно указывает на то, что сопровождаемость продукта вызывает подозрения.

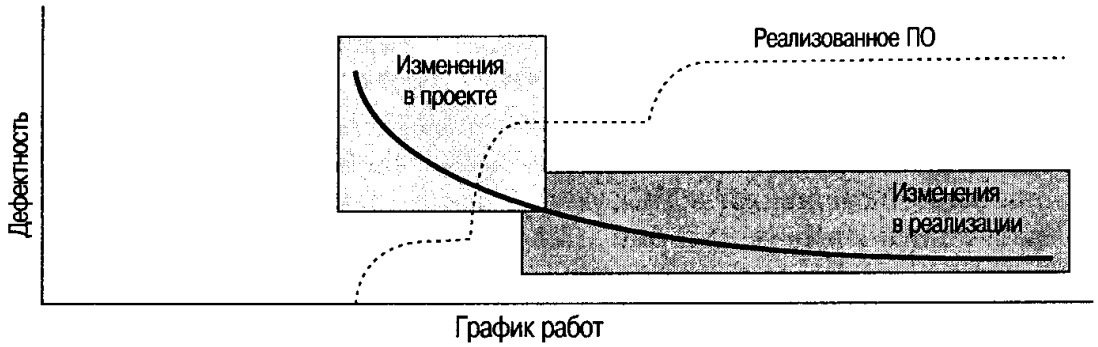


Рис. 13.6. *Ожидаемый коэффициент дефектности на протяжении жизненного цикла благополучного проекта*

13.3.3 Доработки и адаптируемость

Доработка определяется как средняя стоимость внесения изменений, в которую входят затраты на анализ, принятие решения, повторное тестирование всех изменений в основе ПО. *Адаптируемость* определяется как тенденция изменения количества доработок в зависимости от времени. Для благополучного проекта ожидаемая тенденция — уменьшение или стабильность (см. рис. 13.7).

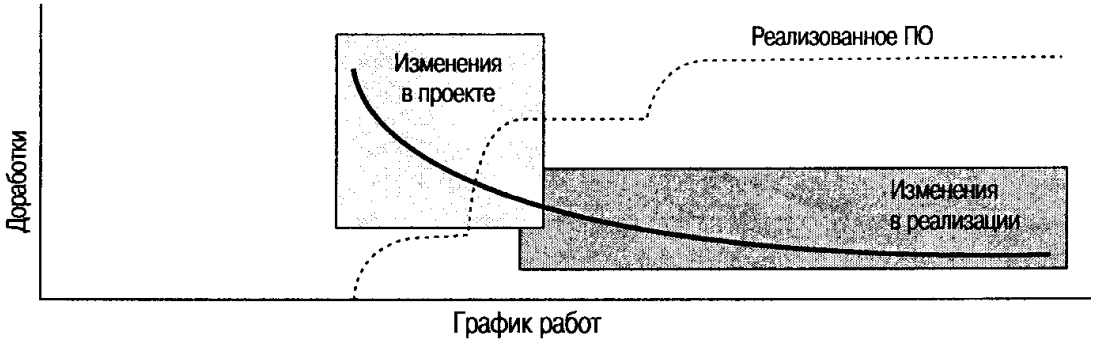


Рис. 13.7. *Ожидаемая адаптируемость на протяжении жизненного цикла благополучного проекта*

Не все изменения одинаковы. Некоторые из них могут быть внесены за один человеко-час, другие требуют нескольких человеко-недель. Эта метрика позволяет измерять объем доработок. В сформировавшемся

итерационном процессе изменения на ранних этапах (изменения в архитектуре, которые оказывают влияние на большое количество компонентов и людей), как правило, требуют больших доработок, чем на поздних (изменения при разработке, которые обычно ограничиваются одним компонентом или человеком). Тенденция к увеличению доработок с течением времени ясно указывает на то, что сопровождаемость продукта вызывает подозрения.

13.3.4 MTBF и завершенность

MTBF — это среднее время использования ПО между двумя отказами. В общем случае MTBF можно вычислить, поделив время тестирования на количество SCO типов 0 и 1. *Завершенность* определяется как тенденция изменения MTBF с течением времени (см. рис. 13.8).

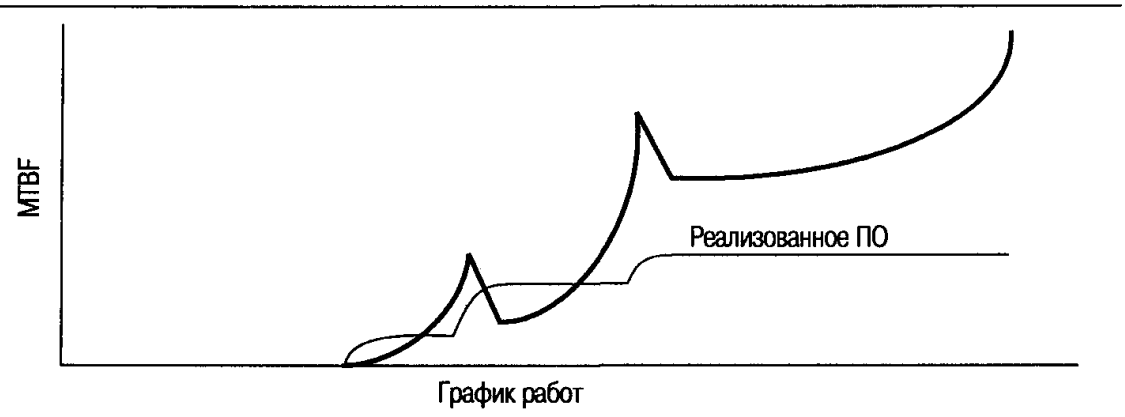


Рис. 13.8. *Ожидаемая завершенность на протяжении жизненного цикла благополучного проекта*

Раннее определение завершенности требует создания эффективной инфраструктуры для тестирования. Традиционные подходы к тестированию монолитных компьютерных программ уделяют основное внимание достижению полного тестового покрытия каждой строки кода, каждой ветви программы и т.д. В современных распределенных и разбитых на компоненты программных системах такое исчерпывающее тестовое покрытие достижимо только для отдельных компонентов. Системы компонентов более эффективно тестируются с использованием статистических методов. Соответственно параметр «завершенность» определяет статистику за время использования, а не охватывает весь продукт.

Ошибки в ПО могут быть разбиты на две категории: детерминированные и недетерминированные. Физики назвали бы их соответственно ошибками Бора и ошибками Гейзенберга. Ошибки Бора — это класс ошибок, которые всегда проявляются, когда ПО используется определенным образом. Такие ошибки в большинстве случаев являются следствием ошибок кодирования, а необходимые изменения обычно ограничиваются одним компонентом. Ошибки Гейзенберга — это отказы ПО, которые происходят случайно с некоторой вероятностью при наступлении некоторой ситуации. Эти ошибки почти всегда возникают из-за ошибок,

допущенных при разработке (и зачастую требующих внесения изменений в множество компонентов), и обычно не воспроизводятся даже в тех случаях, когда ПО используется одним и тем же видимым способом. Для обеспечения адекватного тестового покрытия и исправления статистически значимых ошибок Гейзенберга требуется экстенсивное статистическое тестирование по реальным и случайным сценариям применения.

Традиционные программные продукты, в которых единственная программа выполняется на единственном процессоре, как правило, содержат только ошибки Бора. Современные распределенные системы с множеством взаимодействующих компонентов, выполняющихся в сети процессоров, оказываются уязвимыми для ошибок Гейзенберга, которые сложно обнаруживать, анализировать и исправлять. Лучший способ сделать программный продукт совершенным заключается в создании с самого начала такой инфраструктуры тестирования, которая допускает выполнение случайных сценариев использования на ранних стадиях жизненного цикла и постоянно увеличивает ширину и глубину сценариев для охвата критичных по надежности компонентов.

По мере своего создания базовые версии ПО должны постоянно проверяться с помощью сценариев тестирования. На основе этого тестирования можно определить параметры надежности. Осмысленная оценка завершенности продукта может быть выполнена за счет максимизации времени тестирования (с использованием независимой среды тестирования, автоматизированных регрессионных тестов, случайного статистического тестирования, тестирования в часы, следующие за большими нагрузками, и т.д.). Такой подход предоставляет мощный механизм для автоматизации тестирования на самых ранних стадиях жизненного цикла. Этот метод может быть применен также для мониторинга производительности и измерения надежности.

13.4 ОЖИДАЕМОЕ ПОВЕДЕНИЕ НА ПРОТЯЖЕНИИ ЖИЗНЕННОГО ЦИКЛА

Не существует никакого математического или формального обоснования использования именно упомянутых выше семи метрик. Однако их выбор обусловлен особыми причинами:

- Показатели качества определяются на основе состояния самого продукта, а не по отдельным рабочим продуктам.
- Они позволяют выявить ненужные затраты, производимые в рамках процесса. Метрики дефектов и доработки — это стандартные меры для большинства производственных процессов.
- Они соответствуют динамическому характеру, присущему итерационной разработке. Вместо того чтобы уделять повышенное внимание абсолютному значению, они явно концентрируются на тенденциях или изменениях в зависимости от времени.
- Текущие значения и текущие тенденции изменения метрик используются в качестве параметров управления проектом.

Реальные значения этих метрик могут изменяться в широких пределах в рамках проектов, организаций и областей применения. Однако относительные тенденции изменения в течение стадий проекта должны следовать общим примерам, приведенным в таблице 13.3. Солидная организация-разработчик должна уметь описывать те значения метрик, к которым она стремится, что является более информативным и точным для ее основной деятельности и конкретных процессов.

Таблица 13.3.
Стандартные образцы изменения метрик на протяжении жизненного цикла

Метрика	Начальная стадия	Уточнение	Конструирование	Ввод в действие
Прогресс	5%	25%	90%	100%
Архитектура	30%	90%	100%	100%
Приложения	<5%	20%	85%	100%
Затраты	Низкие	Умеренные	Высокие	Высокие
Работа	5%	25%	90%	100%
Время	10%	40%	90%	100%
Штатное расписание	Небольшая команда	Возрастающее	Устойчивое	Изменчивое
Стабильность	Изменчивая	Умеренная	Умеренная	Высокая
Архитектура	Изменчивая	Умеренная	Стабильная	Стабильная
Приложения	Изменчивая	Изменчивая	Умеренная	Стабильная
Коэффициент дефектности	50% — 100%	25% — 50%	<25%	5% — 10%
Архитектура	>50%	>50%	<15%	<5%
Приложения	>80%	>80%	<25%	<10%
Адаптируемость	Различная	Различная	Качественная	Качественная
Архитектура	Различная	Умеренная	Качественная	Качественная
Приложения	Различная	Различная	Умеренная	Качественная
Завершенность	Прототип	Незначительная	Применимая	Полная
Архитектура	Прототип	Применимая	Полная	Полная
Приложения	Прототип	Незначительная	Применимая	Полная

13.5 ПРАКТИЧЕСКОЕ ИСПОЛЬЗОВАНИЕ МЕТРИК ПО

Всякие измерения полезны, но тем, кто принимает решения, все равно необходимо думать. Измерения только дают данные, которые помогают правильно ставить вопросы, понимать контекст и принимать объективные решения. Поскольку природа проектов по созданию ПО чрезвычайно динамична, то возможность выполнять такие измерения должна

существовать в любое время и быть применима к различным частям изменяющегося продукта (версия, компонент, класс). Измерения должны производиться таким образом, чтобы можно было оценить тенденции (зависимость первой и второй производных от времени). На практике такая ситуация достижима только в тех проектах, где поддерживается доступ в онлайновом режиме к автоматически определяемым метрикам как к побочному продукту среды разработки/интеграции.

Хорошая метрика обладает следующими основными характеристиками:

1. *Она имеет смысл для заказчика, менеджера и исполнителя.* Если хотя бы одна из заинтересованных сторон не рассматривает метрику как осмысленную, она не будет использоваться. «Покупатель всегда прав» — это девиз торговли, а не догмат разработки. Заказчики обращаются к разработчикам, поскольку те — в отличие от заказчиков — являются экспертами в области разработки и управления проектами ПО. Заказчики должны принимать те метрики, которые будут иметь смысл для разработчика.

2. *Она показывает количественную корреляцию между изменениями в процессе и ходом бизнеса.* Единственными реальными целями и задачами организации являются финансовые: снижение затрат, повышение доходов и увеличение прибыли.

3. *Она объективна, и ее определение недвусмысленно.* Объективность следует трансформировать в некоторую форму числового представления (например, цифры, проценты, доли) в противоположность представлению текстовому (например, отличный, хороший, правильный, плохой). Неопределенность минимизируется посредством использования понятных единиц измерения (таких, как человеко-месяцы, SLOC, изменения, функциональные точки, классы, сценарии, требования), которые не так просто определить точно.

4. *Она показывает тенденции.* Это очень важная характеристика. Понимание изменений значения метрики в зависимости от времени, от проекта к проекту, от версии к версии и т.д. оказывается чрезвычайно важным, особенно для современных моделей итерационной разработки. Редко случается, чтобы некоторая метрика напрямую приводила к выполнению каких-либо конкретных действий. Скорее, метрика представляет взгляд с некоторой точки зрения. И интерпретировать эту метрику, и определить, какие действия необходимы, является задачей тех, кто ответственен за принятие решений (менеджера, команды или иной структуры, занимающейся обработкой информации).

5. *Она является естественным побочным продуктом процесса.* Ради метрики не добавляются новые рабочие продукты или дополнительные виды деятельности; она выводится непосредственно из самих рабочих процессов.

6. *Она может быть автоматизирована.* Опыт показывает, что наиболее удачными метриками являются те, информация о которых собирается и представляется автоматически. Отчасти это так, потому что программный инструментарий требует строгих определений для обрабатываемых данных.

Когда метрики указывают на некоторую проблему, важно вникнуть во все симптомы, чтобы поставить диагноз. Метрики обычно отображают эффекты; для понимания причин необходим синтез многих точек зрения и умозаключений. Например, умозаключения необходимы для правильной интерпретации следующих ситуаций:

- Малое количество запросов на внесение изменений может означать, что ПО совершенно и свободно от ошибок, а может означать то, что команда, ответственная за тестирование, находится в отпуске.
- Запрос на внесение изменений, который остается открытым в течение длительного времени, может означать, что ошибка была легко диагностирована, но ее исправление потребовало значительных доработок, либо может означать, что ее диагностика заняла очень много времени, а исправление заключалось во внесении изменений в одну единственную строку кода.
- Большой рост персонала в текущем месяце может привести к пропорциональному увеличению прогресса, если новый персонал состоит из обученных людей, которые сразу же приступят к производительному труду. И он же может вызвать замедление прогресса, если наняты необученные люди, которым потребуется самая разнообразная поддержка со стороны тех, кто занят производительным трудом, для ввода в курс дела.

Качественные оценки не могут выполняться самими метриками, для этого требуются интеллектуальные усилия со стороны, например, менеджеров проектов по созданию ПО.

13.6 АВТОМАТИЗАЦИЯ МЕТРИК

Существует множество возможностей автоматизации деятельности по контролю за проектом. Основным средством управления в соответствии с планом является панель управления проектом (Software Project Control Panel, SPCP), которая поддерживает доступ к текущему состоянию рабочих продуктов в онлайновом режиме. Такой подход впервые был рекомендован Airlie Software Council [Brown, 1996], который использовал метафору «приборной доски» (dashboard) проекта. Идея заключается в том, чтобы вывести на экран дисплея панель, в которой были бы сведены вместе данные, полученные из разных источников, для того чтобы показать текущее состояние какого-либо отдельного аспекта проекта. Например, менеджер проекта может просмотреть общие показатели проекта; менеджеру, отвечающему за тестирование, могут понадобиться данные,

касающиеся параметров предстоящей бета-версии; менеджеров по разработке могут интересовать только данные по подсистемам и компонентам, за разработку которых они отвечают. Панель может иметь некоторые стандартные возможности, вроде предупреждающих цветов, границ, переменных шкал, цифровых форматов и аналоговых форматов для представления текущей ситуации в общем плане. Она также может обеспечивать широкие возможности для подробного анализа ситуации. Такая автоматическая поддержка поможет менеджерам понять прогресс и качественные тенденции, а также способствует выбору метрик командой разработчиков.

Для полной реализации SPCP необходимо определить и разработать:

- Составляющие метрик: показатели, тенденции, сравнения и прогрессии.
- Графический пользовательский интерфейс: GUI-поддержка для лица, выполняющего роль менеджера проекта, гибкие возможности для поддержки других ролей.
- Агенты сбора метрик: извлечение данных из инструментов среды, которые поддерживают различные нотации для разных комплектов рабочих продуктов.
- Сервер управления метриками: поддержка управления данными для наполнения метрик конкретными значениями и хранения данных, извлеченных агентами.
- Определение метрик: реальное представление метрик для прогресса требований (извлекаемых из рабочих продуктов комплекта требований), для прогресса проектирования (извлекаемых из рабочих продуктов комплекта проектирования), для прогресса при реализации (извлекаемых из рабочих продуктов комплекта реализации), для прогресса в оценке (извлекаемых из рабочих продуктов комплекта внедрения) и для других составляющих прогресса (извлекаемых из руководств, систем управления финансами, рабочих продуктов управления и т.д.).
- Действующие лица: обычно наблюдатель и администратор.

В состав специальных наблюдателей (выполняющих определенные роли) входят менеджеры проекта по созданию ПО, руководители команд по разработке ПО, разработчики архитектуры ПО и заказчики. Для каждой роли существуют своя конфигурация панели управления и диапазон представляемых данных. Каждая роль предполагает выполнение одних и тех же общих вариантов использования, но с разными акцентами:

- Наблюдатель: определяет структуру вывода на основе существующих механизмов, графических объектов и привязок к данным проекта; запрашивает данные, которые должны выводиться на различных уровнях абстракции.

- Администратор: инсталлирует систему; описывает новые механизмы, графические объекты и привязки; осуществляет функции по хранению; определяет структуры композиции и декомпозиции для вывода различных уровней абстракции.

Вывод в целом называется панелью. Внутри панели присутствуют графические объекты, которые обеспечивают различные типы графического вывода (например, круговые и прямоугольные диаграммы) информации. Каждый графический объект отображает одну метрику. Панель обычно содержит некоторое число графических объектов с конкретной геометрией расположения. Параметр, описываемый данным графическим объектом, маркируется типом метрики, итоговым уровнем и именем экземпляра (таким, как число строк кода, подсистема, сервер 1). Метрики могут выводится в двух режимах: в виде значения, относящегося к конкретному моменту времени, либо в виде графика, отображающего множественные и последовательные моменты времени. К графическому изображению метрик применимы лишь некоторые типы вывода.

Метрики могут выводиться с контрольными значениями или без них. Контрольное значение — это ожидаемое текущее значение, абсолютное или относительное, которое используется для сравнения с динамически изменяющейся метрикой. Например, планируемое значение некоторой метрики прогресса является контрольным значением для сравнения с реальным значением этого параметра. Другой пример контрольных значений — пороговые значения. Преодоление порогового значения может приводить к такому изменению состояния, которое будет очевидно для пользователя. Контрольные значения могут выводиться в рамках того же графического объекта, что и соответствующая метрика, чтобы пользователь мог их визуально сравнивать.

Показатели могут представлять данные в различных форматах: двоичном (например, черный и белый), троичном (например, красный, желтый и зеленый), цифровом (целое или с плавающей точкой) или в некотором другом формате перечислимого типа (т.е. последовательность возможных дискретных значений: вос..суб, готовься-целься-огонь, янв..дек). Показатели также предоставляют механизм, который может быть использован для компактного изображения условия или обстоятельства, связанного с другой метрикой, либо связей между различными метриками, с одной стороны, и соответствующими им контрольными значениями, с другой.

Графическое изображение тенденции показывает зависимость значений от времени и позволяет задавать верхнее и нижнее пороговые значения. Преодоление порогового значения может быть привязано к примечательному изменению состояния соответствующего показателя от зеленого к красному или наоборот. Графики должны выводиться с шагом по времени, выбранным пользователем (например, день, неделя, месяц, квартал, год). На сравнительном графике выводится множество значений одновременно в зависимости от времени. Схождение или расхождение значений может быть привязано к какому-либо индикатору. График прогресса показывает процент завершения, где элементы прогресса

изображены как переходы из одного состояния в другое, и каждому состоянию ставится в соответствие приобретенная стоимость. Графики, отображающие тенденцию, сравнение и прогресс, приведены на рис. 13.9.

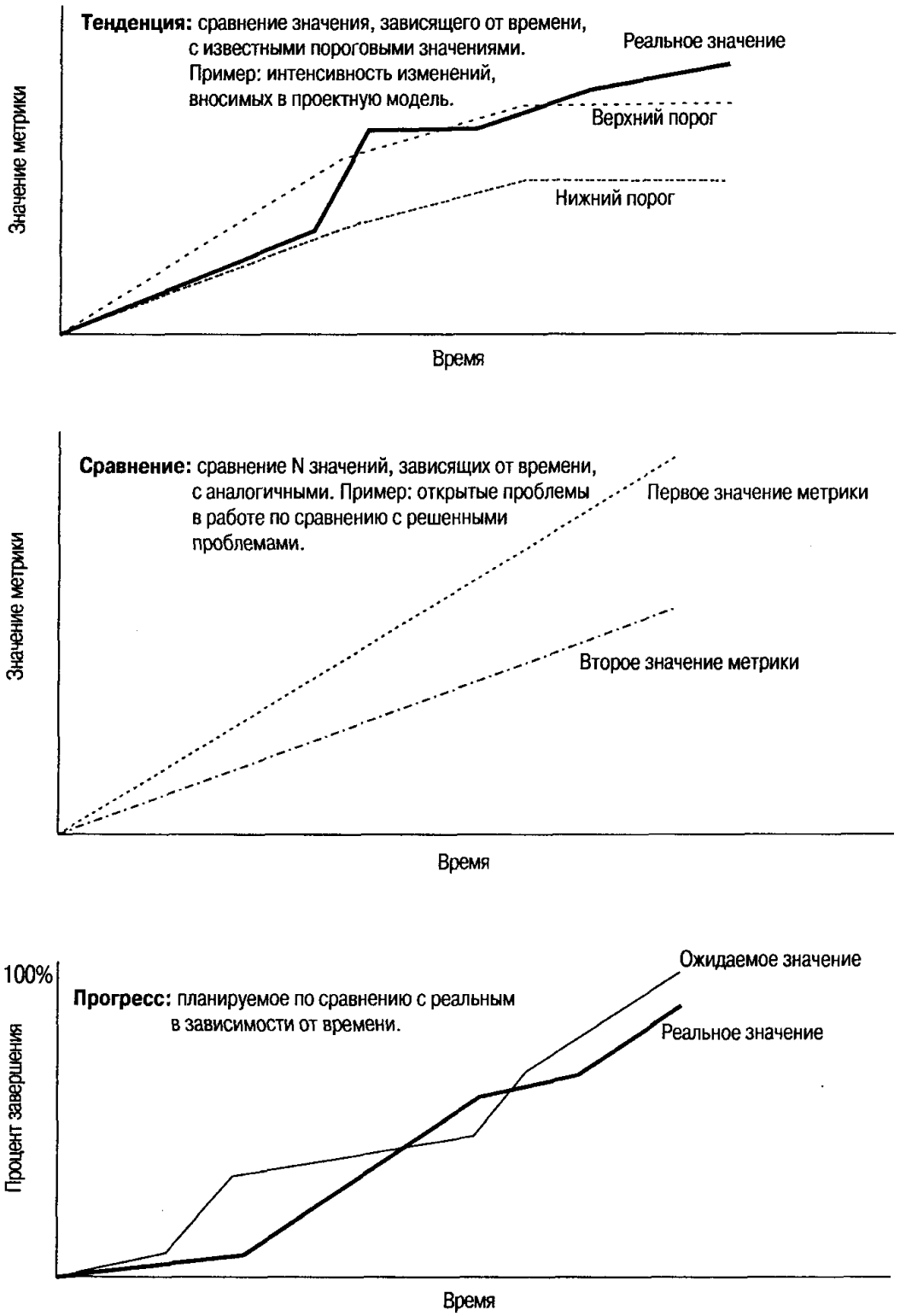


Рис. 13.9. *Примеры классов фундаментальных метрик*

Сведения по данной метрике могут обобщаться по определенной пользователем линейной схеме. (Например, строки кода могут подсчитываться для модуля, подсистемы, проекта.) Проект – это высший уровень обобщения для всех данных, принадлежащих конкретному комплекту (контекст высшего уровня). Пользователи могут задавать структуру обобщения, выбирать выводимый уровень, основываясь на заранее заданных структурах, а также двигаться вниз от обобщенного числа, просматривая детали все более низкого уровня.

На рис. 13.10 показан простой пример SPCP для некоторого проекта. В данном случае для менеджера проекта по созданию ПО определен вывод самого верхнего уровня, содержащий четыре графических объекта.

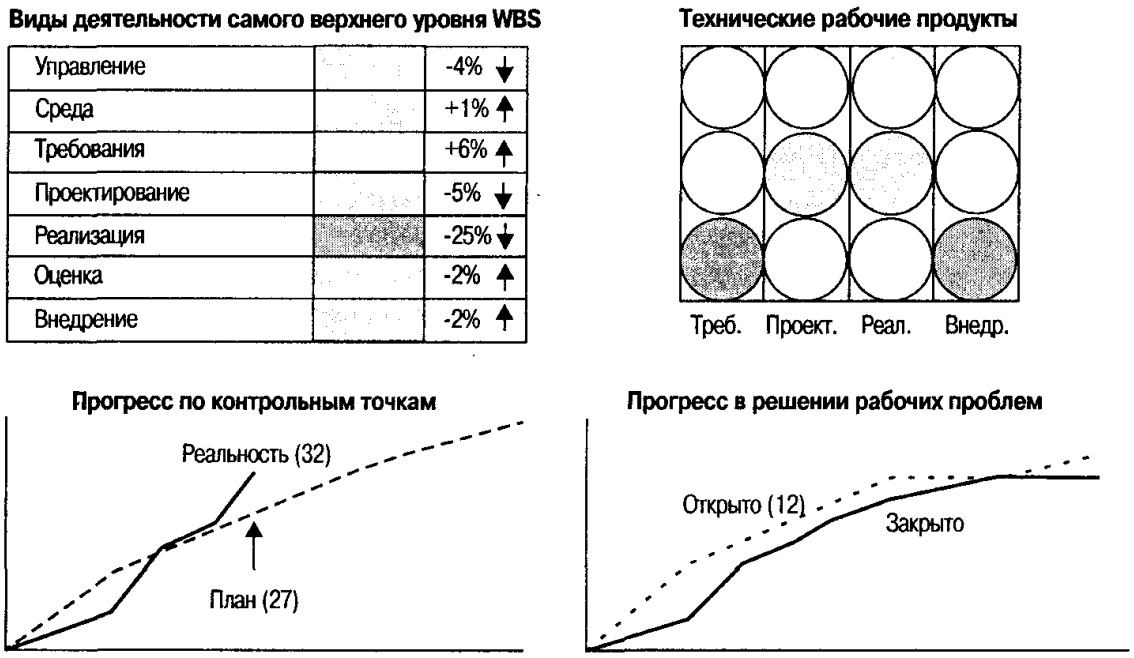


Рис. 13.10. *Пример отображения SPCP для самого верхнего уровня проекта*

1. Состояние работ по проекту. Графический объект, расположенный в верхнем левом углу, дает общее представление о состоянии WBS-элементов самого верхнего уровня. Все семь элементов могут выводиться красным, желтым и зеленым цветом для отображения текущего состояния приобретенной стоимости. (На рис. 13.10 используются белый цвет и оттенки серого.) Например, зеленый цвет будет означать *опережение плана*, желтый цвет – *отклонение от плана не более 10%*, а красный будет выделять элементы, чье *отставание от плана превышает 10%*. Этот графический объект имеет несколько типов показателей: три различных цвета, реальный процент и первую производную на текущий момент (стрелка вверх означает тенденцию к улучшению, стрелка вниз – тенденцию к ухудшению).

2. Состояние технических рабочих продуктов. Графический объект, расположенный в верхнем правом углу, дает общее представление о текущем состоянии технических рабочих продуктов. Цвет кружка «Треб.» показывает оценку текущего состояния моделей вариантов использования и спецификаций требований, цвет «Проект.» — то же самое для проектных моделей, цвет «Реал.» — для исходного кода, а цвет «Внедр.» — для тестирования программы.
3. Прогресс по контрольным точкам. Графический объект, расположенный в нижнем левом углу, выдает оценку прогресса в достижении контрольных точек по сравнению с запланированным и приводит показатели текущих значений.
4. Прогресс в решении рабочих проблем. Графический объект, расположенный в нижнем правом углу, дает оценку прогресса с другой точки зрения, показывая текущее число открытых и решенных проблем.

Рис. 13.10 является одним из примеров возможной реализации метрики, связанной с прогрессом. Этот пример тривиален, но он дает представление об основных возможностях вывода SPCP. Формат и содержимое любой панели проекта могут быть настроены в соответствии с предпочтениями менеджера проекта по созданию ПО для отслеживания параметров, представляющих интерес на самом верхнем уровне. Некоторые менеджеры, возможно, пожелают, чтобы на экране выводились только обобщенные данные и некоторые ключевые тенденции. Другим потребуется большое количество тенденций и конкретных деталей. SPCP должна быть настраиваемой и предоставлять возможности для получения детальной информации по любой метрике. Например, требование красного цвета для изображения рабочих продуктов внедрения может приводить к получению следующего уровня детализации либо во времени (диаграмму тенденций), либо в абстракции (подробное положение дел с тестированием для каждой версии, подсистемы и т.д.).

Следующий вариант использования самого верхнего уровня, который описывает основное рабочее понятие для SPCP, соответствует случаю взаимодействия наблюдателя с панелью управления:

- Запуск SPCP. SPCP запускается и выводит основную текущую информацию, которая сохранилась с момента работы с SPCP последнего пользователя.
- Выбор варианта панели. Пользователь выбирает вариант панели из списка ранее определенных стандартных вариантов панели. SPCP выводит выбранный вариант.
- Выбор представления метрики в виде числового значения или графика. Пользователь выбирает, будет ли метрика выводиться для заданного момента времени или в виде графика как тенденция. По умолчанию для значений выводится самое последнее из доступных измерение метрики. Значение по умолчанию для тенденций — ежемесячно.

- Выбор наложения контрольных значений. Пользователь указывает графический объект и запрашивает совместный вывод контрольных значений для этой метрики и значения для конкретного момента времени. В случае тенденций контрольные значения накладываются при выводе на метрику.
- Переход к тенденции. Пользователь указывает графический объект, отображающий значение в некий момент времени, и переходит к просмотру тенденции изменения данной метрики.
- Переход к значению, соответствующему определенному моменту времени. Пользователь выбирает графический объект, показывающий тенденции, и переходит к просмотру значений данной метрики.
- Переход к более низким уровням информации. Пользователь выбирает графический объект, показывающий значение в некий момент времени, и переходит к просмотру информации следующего уровня.
- Переход к более низкому уровню показателей. Пользователь выбирает графический объект с неким показателем и развертывает следующий уровень показателей.

SPCP – это лишь один из примеров подхода к автоматизации метрик, который позволяет собирать, организовывать и выводить значения и тенденции, извлекаемые напрямую из рабочих продуктов. Разработчики ПО принимают метрики только в том случае, если они автоматизированы в данной среде.

Глава 14

Адаптация процесса

Ключевые моменты

- Схема процесса должна подстраиваться под особенности данного проекта.
- Масштаб проекта, в частности размер команды, влияет на конфигурацию процесса в большей степени, чем любые другие факторы.
- Другими ключевыми факторами являются взаимоотношения всех заинтересованных сторон, гибкость процесса, зрелость процесса, архитектурный риск и опыт в данной области.
- Реализация особенностей процесса будет меняться, но дух, лежащий в основе процесса, остается неизменным.

Деятельность по управлению созданием ПО охватывает широкий спектр областей. Несмотря на существование некоторых универсальных подходов и методов, процессу всегда необходима некоторая адаптация к специфическим требованиям конкретного проекта. Разработчик коммерческого программного инструмента, полностью контролирующий структуру своих инвестиций, будет использовать процесс, отличающийся от того, который бы использовал системный интегратор, работающий по контракту, для автоматизации системы безопасности атомной электростанции. Нет никаких сомнений в том, что зрелый процесс и эффективные подходы к управлению созданием ПО принесут гораздо больше пользы интегратору крупномасштабного ПО, чем разработчику небольших инструментов. Тем не менее в зависимости от целей бизнеса отдача от вложенных ресурсов, получаемая при применении более совершенных подходов к управлению созданием ПО, окажется стоящей для любой организации-производителя ПО.

14.1 ДИСКРИМИНАНТЫ ПРОЦЕССА

При адаптации процесса управления к конкретной области или проекту существуют два класса определяющих факторов: техническая сложность и сложность управления. На рис. 14.1 приведены эти два измерения процесса и показаны примеры в приложении к проектам. Формализм рассмотрения, контроль качества рабочих продуктов, приоритеты подходов

и другие многочисленные параметры, характеризующие процесс, зависят от того, какое место занимает проект в этих двух измерениях. Рис. 14.2 обобщает различные приоритеты по данным измерениям.

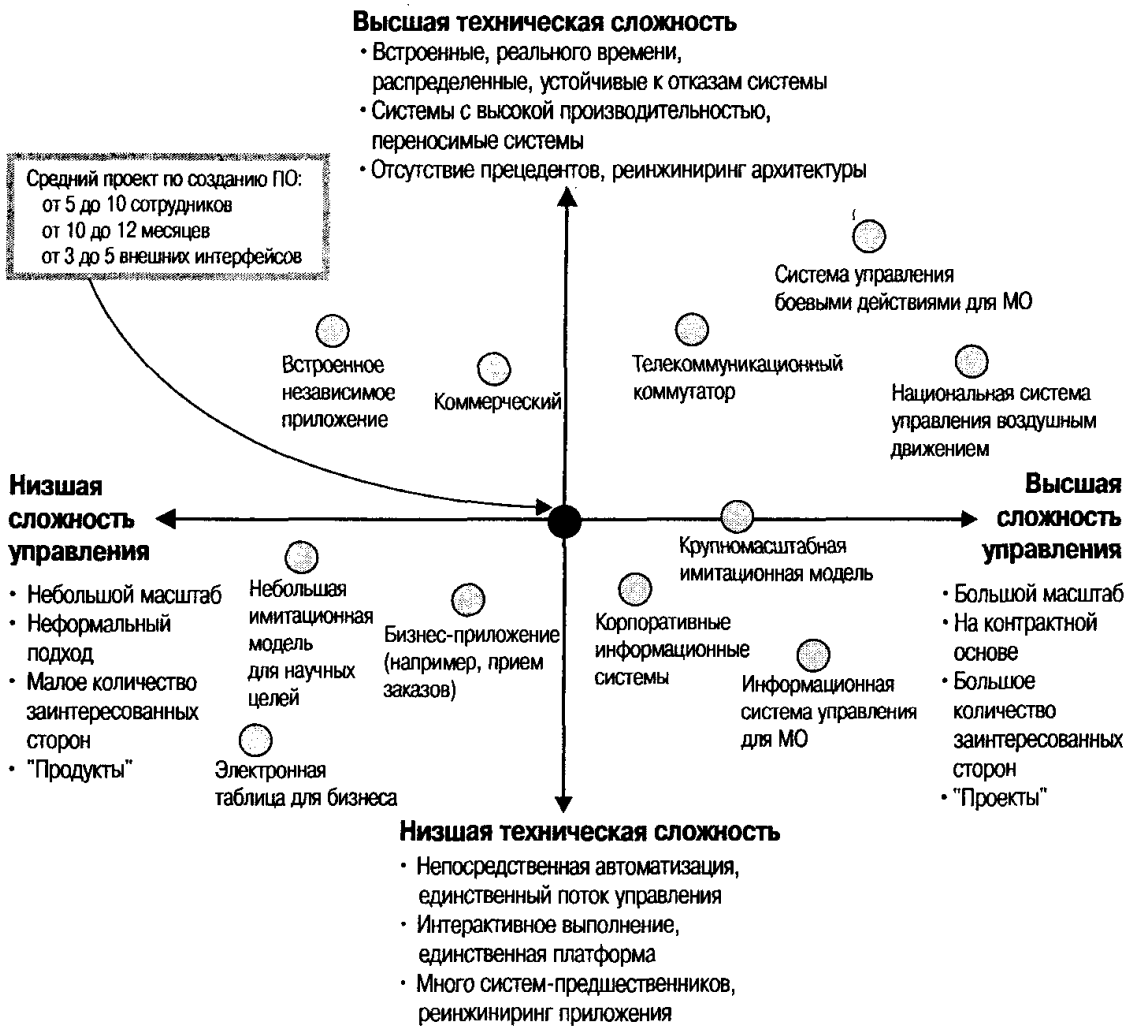


Рис. 14.1. *Два главных измерения изменчивости процесса*

Схема процесса *не является* реализацией процесса для конкретного проекта с четко определенным рецептом успеха. Во-первых, потребуется рассудительность, а, во-вторых, методы, способы, культура, формализация и организация должны быть адаптированы под конкретную область для достижения такой реализации процесса, которая приведет к успеху. Следующее ниже обсуждение основных различий в процессах выполнения проектов строится вокруг шести параметров процесса: размера проекта и пяти параметров, которые влияют на экспоненту процесса и, следовательно, на экономию при больших масштабах в модели COCOMO II. Это лишь некоторые из критичных измерений, которые менеджер проекта по созданию ПО должен принимать во внимание при адаптации схемы процесса для его практической реализации.

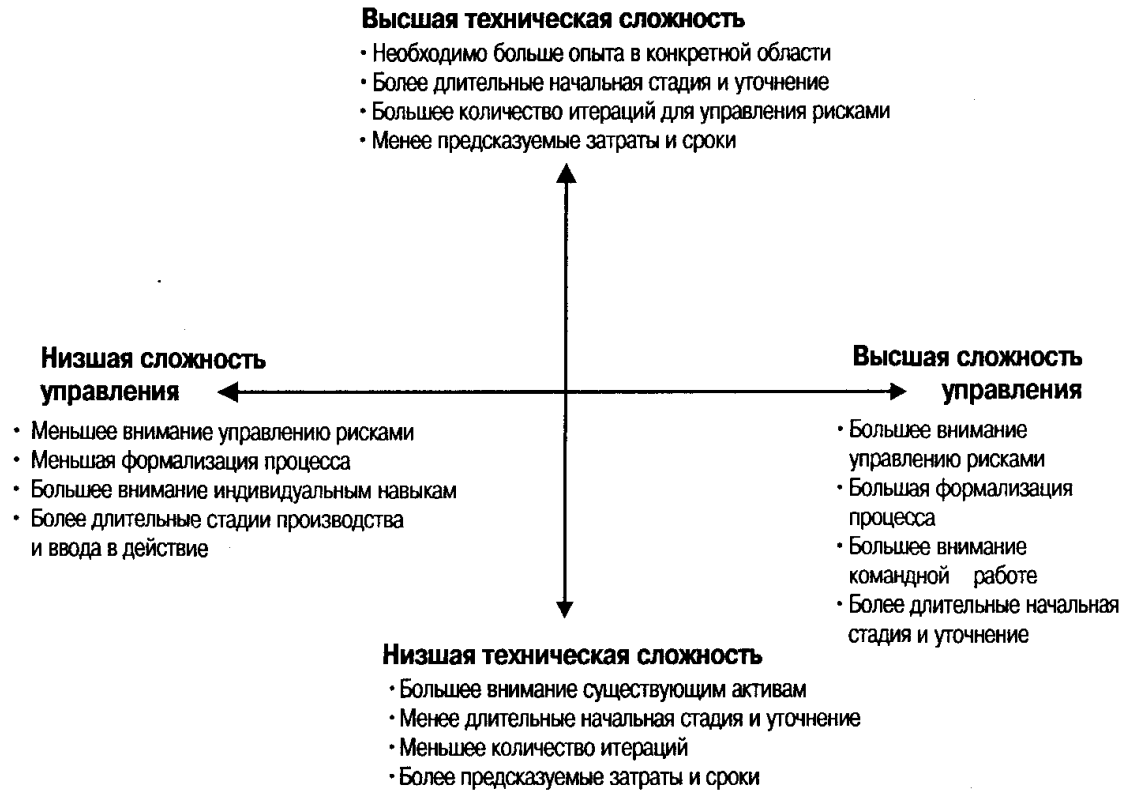


Рис. 14.2. *Приоритеты для адаптации схемы процесса*

14.1.1 Масштаб

Возможно, наиболее важным фактором при адаптации схемы процесса по созданию ПО к конкретным требованиям проекта является общий масштаб приложения. Существует множество способов измерения масштаба, включая количество строк исходного кода, количество функциональных точек, количество вариантов использования и количество долларов. С точки зрения адаптации процесса самым главным критерием масштаба является размер команды. По мере роста числа разработчиков важность согласованного межличностного взаимодействия становится первостепенной. В противном случае плата за большой масштаб может оказать серьезное влияние на достижение целей проекта.

Мой опыт участия в проектах показал, что оптимальный размер команды разработчиков составляет пять человек. В ходе проведения множества исследований было установлено, что большинство людей способны управлять четырьмя-семью объектами одновременно. При простой экстраполяции этих результатов можно предположить, что необходимы фундаментально различающиеся подходы к управлению при руководстве командой из 1 человека (тривиальной), из 5 человек (маленькой), из 25 человек (средней), из 125 человек (большой), из 625 (гигантской) и т.д. По мере роста величины команды каждый новый уровень управления персоналом достигается при каждой очередной степени 5. Эта модель может использоваться для описания некоторых различий в процессах для проектов разной величины.

Проектам тривиального размера не требуется никакой управленческой надстройки (планирование, взаимодействие, координация, оценка прогресса, проверка, администрирование). Практически отсутствует необходимость в документировании промежуточных рабочих продуктов. Все рабочие процессы состоят из единственного потока. Ход выполнения в большой степени зависит от личных навыков.

Маленький проект (5 человек) требует очень небольшой управленческой надстройки, но наличие руководства, направляющего команду на достижение общей цели, является критичным. Необходимо распространять среди членов команды промежуточные рабочие продукты. Контрольные точки проекта легко поддаются планированию, легко достигаются и изменяются. В проекте существует незначительное число индивидуальных рабочих процессов. Ход выполнения зависит прежде всего от личных навыков. Зрелость процесса относительно не важна. Индивидуальные инструменты могут иметь решающее влияние на ход выполнения.

Проекты среднего размера (25 человек) требуют умеренной управленческой надстройки, включая ответственного менеджера проекта для синхронизации различных рабочих процессов команды и для равномерного распределения ресурсов. В дополнительные обязанности лидеров команд входят проверка, координация и оценка. Существует определенная необходимость в распространении промежуточных рабочих продуктов среди команд. Контрольные точки проекта планируются и достигаются формально, а влияние изменений обычно является доброкачественным. Число параллельных командных рабочих процессов невелико; каждый из них состоит из нескольких индивидуальных рабочих процессов. Ход выполнения сильно зависит от навыков ключевых сотрудников, особенно лидеров команд. Весьма ценной становится зрелость процесса. Существенное влияние на ход работ может оказать среда, однако успех может быть достигнут за счет использования определенных инструментов.

Большие проекты (125 человек) требуют значительной управленческой надстройки, включая ответственного менеджера проекта и нескольких менеджеров более низкого уровня для синхронизации различных рабочих процессов уровня проекта и последующего уровня, а также для равномерного распределения ресурсов. Увеличивается объем дополнительной работы, которая ложится на лидеров команд и которая направлена на распределение, проверку, координацию и оценку. Совершенно очевидной становится необходимость распространения промежуточных рабочих продуктов среди разнообразных команд для согласования результатов разработки. Контрольные точки проекта планируются и достигаются формально, а внесение изменений в планы контрольных точек оказывается очень дорогим. Требуется наличие большого числа параллельных командных рабочих процессов, каждый из которых состоит из множества индивидуальных рабочих процессов. На ход выполнения большое влияние оказывают навыки ключевых сотрудников, особенно менеджеров второго уровня и лидеров команд. Выполнение проекта зависит от средних сотрудников по двум причинам:

1. Любой большой проект включает в себя определенный объем рутинного труда, особенно вспомогательные рабочие процессы.
2. Мала вероятность того, что удастся нанять, содержать и удержать большое число выдающихся работников.

Зрелость процесса становится необходимой; это касается, в частности, аспектов планирования и контроля при управлении проектом, прогресса и ожиданий заинтересованных сторон. Интегрированная среда должна управлять внесением изменений, автоматизировать создание рабочих продуктов и поддерживать согласованность изменяющихся рабочих продуктов.

Гигантские проекты (625 человек) требуют существенной управленческой надстройки, включая нескольких менеджеров проекта по созданию ПО и множество менеджеров более низкого уровня для различных рабочих процессов уровня проекта и последующих уровней, а также для равномерного распределения ресурсов. Возрастают затраты на дополнительную работу для лидеров команд, направленную на распределение, проверку, координацию и оценку. Совершенно очевидной становится необходимость распространения промежуточных рабочих продуктов среди разнообразных команд. Контрольные точки проекта планируются и достигаются весьма формально, а внесение изменений в планы контрольных точек обычно приводит к проблемам и к необходимости повторного планирования. Существует большое число параллельных командных рабочих процессов, каждый из которых состоит из множества индивидуальных рабочих процессов. Ход выполнения сильно зависит от навыков ключевых сотрудников, особенно менеджеров второго уровня и лидеров команд, а также от навыков средних сотрудников.

Зрелость процесса создания ПО и наличие опыта в данной области необходимы для исключения рисков и гарантированной синхронизации ожиданий многих заинтересованных сторон. Всем командам разработчиков требуется развитая интегрированная единая среда, позволяющая управлять изменениями, автоматизировать создание рабочих продуктов, поддерживать согласованность изменяющихся рабочих продуктов и увеличивать отдачу от инвестиций с помощью общих процессов, инструментов, нотаций и метрик.

В таблице 14.1 обобщены некоторые существенные различия между составляющими малых и больших проектов.

Таблица 14.1.
Отличительные особенности процессов, определяемые их размерами

Составляющая процесса	Меньший размер команды	Больший размер команды
Стадии жизненного цикла	Размытые границы между стадиями	Четко определенный переход от стадии к стадии для синхронизации прогресса, достигнутого при выполнении параллельных работ

Таблица 14.1. (продолжение)
Отличительные особенности процессов, определяемые их размерами

Составляющая процесса	Меньший размер команды	Больший размер команды
Рабочие продукты	Основное внимание уделяется техническим рабочим продуктам Небольшое количество различных базовых версий Требуется очень небольшое количество рабочих продуктов управления	Управление внесением изменений в технические рабочие продукты, что может привести к множеству базовых версий Большая значимость рабочих продуктов управления
Распределение усилий по различным рабочим процессам	В большей степени требуются разносторонние сотрудники, люди, которые могут принимать участие в различных рабочих процессах	Более высокий процент специалистов Большее число людей и команд, которые уделяют внимание конкретному рабочему процессу
Контрольные точки	Большое количество неформальных событий, используемых для поддержания технической согласованности Отсутствие срывов графика	Небольшое количество формальных событий Синхронизация между командами, которая может длиться несколько дней
Дисциплина управления	Неформальное планирование, контроль за ходом проекта и организация	Формальное планирование, контроль за ходом проекта и организация
Дисциплина автоматизации	Большее количество узкоспециализированных видов среды для работы отдельных сотрудников	Инфраструктура, гарантирующая доступность согласованной среды для всех команд Дополнительная интеграция инструментов для поддержки контроля над проектом и изменениями

14.1.2 Сотрудничество или соперничество заинтересованных сторон

Степень сотрудничества и координации между заинтересованными сторонами (покупателями, разработчиками, пользователями, субподрядчиками, осуществляющими сопровождение и др.) может существенно влиять на особенности определения процесса. Взаимоотношения могут меняться от сплоченности до вражды. У сплоченных команд — общие цели, взаимодополняющие навыки и тесные связи. У враждующих команд — противоречивые цели, конкурирующие или неполные навыки и слабое взаимодействие.

Для продукта, который финансируется, разрабатывается, продвигается на рынок и продается в рамках одной и той же организации, может быть сформулирована единая цель (например, получение прибыли). Внутри маленькой, компактной организации может быть установлен такой порядок, при котором существуют единая база навыков и постоянное взаимодействие между членами команды.

Более трудной задачей является организация большой контрактной работы без наличия соперничества между командами. Подрядчик редко обладает всеми необходимыми знаниями в программной или предметной области и зачастую вынужден объединяться с множеством субподрядчиков, которые имеют конкурентные цели по получению прибыли. Ответственные за финансирование и пользователи стремятся минимизировать стоимость, максимизировать набор возможностей и ускорить выход на рынок, в то время как получившие контракт на разработку стремятся максимизировать прибыль. Большие команды практически невозможно правильно расставить, и синхронизация ожиданий заинтересованных сторон является постоянной проблемой. Все эти факторы способствуют деградации сплоченности команды, и с этим приходится постоянно работать.

В таблице 14.2 обобщены различия в составляющих процесса для разных уровней сплоченности заинтересованных сторон.

Таблица 14.2.
Отличительные особенности, проявляющиеся при разной степени сплоченности заинтересованных сторон

Составляющая процесса	Меньшее количество заинтересованных сторон, сплоченные команды	Большое количество заинтересованных сторон, враждебные отношения
Стадии жизненного цикла	Слабая связь между стадиями	Четко определенный переход от стадии к стадии для синхронизации прогресса, достигнутого при выполнении параллельных работ
Рабочие продукты	Меньшее количество и меньшая детализация рабочих продуктов управления	Первостепенная значимость рабочих продуктов управления, в особенности бизнес-плана, общей концепции и оценки состояния
Распределение усилий по различным рабочим процессам	Меньшие затраты на оценку	Большие затраты на оценку как гарантия согласия заинтересованных сторон
Контрольные точки	Большое количество неформальных событий	Три-четыре формальных события Большое количество неформальных технических проходов, необходимых для синхронизации технических решений Синхронизация между заинтересованными сторонами, которая может задерживать дальнейший прогресс на несколько недель
Дисциплина управления	Неформальное планирование, контроль за ходом проекта и организация	Формальное планирование, контроль за ходом проекта и организация
Дисциплина автоматизации	(не имеет значения)	Необходимость онлайновой среды для каждой заинтересованной стороны

14.1.3 Гибкость или жесткость процесса

Степень жесткости, формализации и простоты внесения изменений, присущая конкретному «соглашению» о проекте (общей концепции, бизнес-плану и плану разработки) оказывает существенное влияние на выполнение проекта. Управление «свободными» контрактами, такими как создание коммерческого продукта внутри структурного подразделения компании, производящей ПО (например, приложения компании Microsoft или инструменты для разработки корпорации Rational Software), имеет минимальную степень сложности. Для процессов разработки такого рода набор функциональных возможностей, время выхода на рынок, бюджет и качество могут быть легко согласованы и изменены с малыми накладными расходами. Например, если компания принимает решение об исключении нескольких возможностей из продукта, находящегося в процессе разработки, для захвата доли рынка за счет ускорения выпуска продукта, вполне допустимым будет принятие решения менее чем за неделю. Все усилия по координации вполне могут ограничиться соглашением по некоторым ключевым вопросам между менеджером по разработке, менеджером по маркетингу и менеджером структурного подразделения.

С другой стороны, в случае жесткого контракта может понадобиться много месяцев на утверждение тех или иных изменений в графике выпуска версий. Например, для того чтобы избежать большого объема разработок на заказ, нужно включить в очередное поколение системы управления воздушным движением некоторый новый коммерческий продукт. Такого рода изменения потребуют согласования с подрядчиком разработки, финансирующим органом, пользователями (возможно, с профсоюзом диспетчеров воздушного сообщения и основными авиакомпаниями), сертифицирующими органами (такими, как Федеральная администрация по авиации), смежными подрядчиками для утверждения интерфейсов взаимодействующих систем и др. Для крупномасштабных, критичных к отказам систем характерны всеобъемлющая жесткость контрактов и использование существенно отличных подходов к управлению. В таблице 14.3 обобщены основные различия в составляющих процесса для разных уровней его гибкости.

Таблица 14.3.
Отличительные особенности, проявляющиеся при различиях в степени гибкости процесса

Составляющая процесса	Гибкий процесс	Жесткий процесс
Стадии жизненного цикла	Допустимость спонтанных решений внутри стадий	Необходимость более солидного основания для принятия решения на начальной стадии
Рабочие продукты	Изменяемость бизнес-плана и общей концепции	Тщательно отслеживаемые изменения бизнес-плана и общей концепции

Таблица 14.3. (продолжение)
Отличительные особенности, проявляющиеся при различиях в степени гибкости процесса

Составляющая процесса	Гибкий процесс	Жесткий процесс
Распределение усилий по различным рабочим процессам	(не имеет значения)	Возросшие уровни рабочих процессов управления и оценки
Контрольные точки	Большое количество неформальных событий, направленных на поддержку технической согласованности	Три-четыре формальных события Синхронизация между заинтересованными сторонами, которая может задерживать дальнейший прогресс на дни и недели
Дисциплина управления	(не имеет значения)	Необходимость более качественного планирования и контроля над проектом
Дисциплина автоматизации	(не имеет значения)	(не имеет значения)

14.1.4 Зрелость процесса

Уровень зрелости процесса, присущий данной организации-разработчику, как это определено в модели Capability Maturity Model, предложенной Software Engineering Institute [SEI, 1993; 1993b; 1995], является еще одним ключевым фактором, влияющим на сложность управления. Управлять зрелым процессом (уровень 3 или выше) гораздо проще, чем незрелым процессом (уровни 1 и 2). Организации, обладающие зрелым процессом, обычно располагают более высоким уровнем предшествующего опыта в области разработок ПО и высоким уровнем дополнительного обеспечения, которое способствует предсказуемости в планировании и в осуществлении процесса. Такого рода дополнительное обеспечение включает в себя четко определенные методы, инструменты для автоматизации процесса, обученный персонал, метрики планирования, шаблоны рабочих продуктов и рабочих процессов. Адаптация процесса со зрелой организацией под конкретный проект является, вообще говоря, простой задачей. В таблице 14.4 представлены ключевые различия в составляющих процесса для различных уровней его зрелости.

Таблица 14.4.
Отличительные особенности, проявляющиеся при различиях в степени зрелости процесса

Составляющая процесса	Зрелая организация, уровень 3 или 4	Организация уровня 1
Стадии жизненного цикла	Хорошо проработанные критерии для перехода от одной стадии к другой	(не имеют значения)

Таблица 14.4. (продолжение)
Отличительные особенности, проявляющиеся при различиях в степени зрелости процесса

Составляющая процесса	Зрелая организация, уровень 3 или 4	Организация уровня 1
Рабочие продукты	Хорошо проработанные формат, содержание и методы изготовления	Свободная форма
Распределение усилий по различным рабочим процессам	Хорошо проработанная основа	Основа отсутствует
Контрольные точки	Хорошо описанное сочетание формальных и неформальных событий	(не имеет значения)
Дисциплина управления	Предсказуемое планирование Объективная оценка состояния	Неформальное планирование и контроль над проектом
Дисциплина автоматизации	Требует высоких уровней автоматизации для выполнения «круговой» разработки, управления изменениями и инструментального оснащения процесса	Низкий уровень автоматизации или несвязанные островки автоматизации

14.1.5 Архитектурный риск

Степень технической осуществимости, демонстрируемая перед переходом к полномасштабному производству, является важным измерением при определении специфики процесса в данном проекте. Существует много потенциальных источников архитектурного риска. К наиболее важным и повторяющимся относятся работа системы (использование ресурсов, время реакции, пропускная способность, точность), простота внесения изменений (добавление новых возможностей, включение новой технологии, настройка на динамически изменяющиеся условия работы) и надежность системы (предсказуемое поведение, устойчивость к отказам). Та степень, с которой удается избавиться от этих рисков до начала производства, приводит к разительным отличиям в процессе адаптации. В таблице 14.5 приведены ключевые различия в составляющих процесса для разных уровней архитектурного риска.

Таблица 14.5.
Отличительные особенности, проявляющиеся при различиях в степени архитектурного риска

Составляющая процесса	Демонстрация реализуемости архитектуры в целом	Отсутствие демонстрации реализуемости
Стадии жизненного цикла	Большее число итераций на начальной стадии и стадии уточнении	Меньшее число итераций на начальных стадиях Большее число итераций на стадии конструирования

Таблица 14.5. (продолжение)
Отличительные особенности, проявляющиеся при различиях в степени архитектурного риска

Составляющая процесса	Демонстрация реализуемости архитектуры в целом	Отсутствие демонстрации реализуемости
Рабочие продукты	Широта и глубина всех рабочих продуктов на более ранних стадиях	(не имеют значения)
Распределение усилий по различным рабочим процессам	Более высокий уровень усилий, затрачиваемых на проектирование Более низкий уровень усилий, затрачиваемых на реализацию и оценку	Более высокий уровень усилий, затрачиваемых на реализацию и оценку, из-за возросшего объема дефектов и доработок
Контрольные точки	Большее внимание демонстрациям работы ПО	Большее внимание брифингам, документам и эмуляции
Дисциплина управления	(не имеет значения)	(не имеет значения)
Дисциплина автоматизации	Требует большего количества ресурсов на более ранних стадиях жизненного цикла	Меньшие требования к среде на ранних стадиях жизненного цикла

14.1.6 Опыт в предметной области

Опыт в соответствующей области знаний, которым обладает организация-разработчик, оказывает влияние на способность достижения приемлемой архитектуры за минимальное число итераций. Организация, которая разработала пять поколений коммутаторов, управляющих радарами, в состоянии достичь адекватной базовой архитектуры для нового приложения, связанного с радарами, за две или три итерации по выпуску прототипа. Организации, имеющей большой опыт в создании ПО, но разрабатывающей свое первое приложение, связанное с радарами, может потребоваться четыре или пять версий прототипов, прежде чем она сумеет достичь такого же результата. В таблице 14.6 обобщены ключевые различия в составляющих процесса для разных уровней опыта в предметной области.

Таблица 14.6.
Отличительные особенности, проявляющиеся при различиях в опыте, имеющемся в предметной области

Составляющая процесса	Опытная команда	Неопытная команда
Стадии жизненного цикла	Более короткая стадия разработки	Более длительная стадия разработки
Рабочие продукты	Меньший объем дефектов и доработок в комплектах требований и проектирования	Больший объем дефектов и доработок в комплектах требований и проектирования

Таблица 14.6. (продолжение)
Отличительные особенности, проявляющиеся при различиях в опыте, имеющемся в предметной области

Составляющая процесса	Опытная команда	Неопытная команда
Распределение усилий по различным рабочим процессам	Более низкие уровни усилий, затрачиваемых на требования и проектирование	Более высокие уровни усилий, затрачиваемых на требования и проектирование
Контрольные точки	(не имеют значения)	(не имеют значения)
Дисциплина управления	Меньшее внимание управлению рисками	Требуются менее частые оценки состояния
Требуются более частые оценки состояния	Дисциплина автоматизации	(не имеет значения)

14.2 ПРИМЕР: НЕБОЛЬШОЙ ПРОЕКТ В СРАВНЕНИИ С КРУПНОМАСШТАБНЫМ ПРОЕКТОМ

Анализ отличий между стадиями, рабочими процессами и рабочими продуктами для двух проектов, находящихся на противоположных полюсах спектра по сложности управления, показывает, насколько разными могут быть процессы проектов. Ниже описываются некоторые измерения гибкости, приоритета и качества, которые могут изменяться, если схема процесса применяется к различным приложениям, проектам и предметным областям.

В таблице 14.7 приведены различия в распределении временных затрат по стадиям жизненного цикла в случае большого и малого проекта. Маленькому проекту (например, приложение Windows объемом 50 000 строк исходного кода на Visual Basic, создаваемое командой из 5 человек) могут потребоваться 1 месяц на начальную стадию, 2 месяца на уточнение, 5 месяцев на конструирование и 2 месяца на ввод в действие. Большой, сложный проект (например, бортовая программа для летательного аппарата объемом 300 000 строк исходного кода, создаваемая командой из 40 человек) может потребовать 8 месяцев на начальную стадию, 14 месяцев на уточнение, 20 месяцев на конструирование и 8 месяцев на ввод в действие. Сравнение долей жизненного цикла, приходящихся на каждую стадию, позволяет обнаружить очевидные различия.

Таблица 14.7.
Распределение временных затрат по стадиям для маленьких и больших проектов

	Разработка		Производство	
Предметная область	Начальная стадия	Уточнение	Конструирование	Ввод в действие
Маленький коммерческий проект	10%	20%	50%	20%
Большой, сложный проект	15%	30%	40%	15%

Самым значительным отличием является относительное время, за которое достигается контрольная точка жизненного цикла, связанная с созданием архитектуры. Это соответствует тому количеству времени, которое тратится на стадию разработки, по сравнению с количеством времени, затрачиваемым на стадию производства. Для малых проектов это отношение составляет 30/70; для большого проекта оно ближе к 45/55.

Одним из ключевых аспектов, касающихся различий между двумя такими проектами, является влияние, которое оказывают разные компоненты процесса на успех или неудачу проекта. В этом находит свое отражение важность найма сотрудников или уровень соответствующего управления рисками. Таблица 14.8 содержит перечень рабочих процессов в порядке их важности.

Таблица 14.8.
Различия в приоритетах рабочих процессов для малого и большого проектов

Приоритет	Маленький коммерческий проект	Большой сложный проект
1	Проектирование	Управление
2	Реализация	Проектирование
3	Внедрение	Требования
4	Требования	Оценка
5	Оценка	Среда
6	Управление	Реализация
7	Среда	Внедрение

Следующий список показывает некоторые из ключевых особенностей, являющихся определяющими для успеха. Ни один из этих компонентов процесса нельзя считать незначительным, хотя некоторые из них более важны, чем другие.

- Проектирование — ключевой аспект для обеих областей. Хорошие проектные решения являются для коммерческого проекта ключевым фактором в определении места, занимаемого на рынке, и основой для создания эффективных новых версий продукта. Хорошие проектные решения для большого проекта формируют основу для предсказуемого, эффективного по затратам процесса создания.
- Управление является первостепенным для больших проектов, где ошибки, допущенные при планировании и распределении ресурсов, противоречивые ожидания заинтересованных сторон и другие выводящие из равновесия факторы могут иметь катастрофические последствия для общей динамики работы команд. Для маленькой команды управление не так важно, поскольку вероятность неправильного взаимодействия намного ниже, а его последствия менее значимы.

- Внедрение играет значимую роль для маленького коммерческого продукта, поскольку у него существует широкая пользовательская база и большое разнообразие сред. Единственный в своем роде, сложный проект обычно внедряется только в одном месте. Унаследованные системы и непрерывная эксплуатация могут представлять некоторый риск, однако в общем эти проблемы хорошо понятны и у них постоянный набор целей.

Другой ключевой набор различий присущ реализации рабочих продуктов процесса. В таблице 14.9 приводится концептуальный пример этих различий.

Таблица 14.9.
Различия в рабочих продуктах малого и большого проектов

Рабочий продукт	Маленький коммерческий проект	Большой сложный проект
Декомпозиция работ	Развернутый план, занимающий одну страницу и имеющий 2 уровня элементов WBS	Система финансового управления с 5 или 6 уровнями элементов WBS
Бизнес-план	Подробный план и краткий меморандум	Трехтомные предложения, в том числе технический том, том по затратам и описание соответствующего опыта
Общая концепция	10-страничное общее представление	2200-страничные спецификации подсистем
План разработки	10-страничный план	200-страничный план разработки
Спецификации версий и количество версий	Спецификации трех промежуточных версий	Спецификации 8 — 10 промежуточных версий
Описание архитектуры	5 критичных вариантов использования, 50 диаграмм UML, 20 страниц текста, другие графические материалы	25 критичных вариантов использования, 200 диаграмм UML, 100 страниц текста, другие графические материалы
ПО	50 000 строк кода на Visual Basic	300 000 строк кода на C++
Описание версии	10-страничные замечания по версии	100-страничное обобщение результатов
Внедрение	Курсы обучения пользователей Стандартный набор для продаж	План ввода в действие План инсталляции
Руководство пользователя	Онлайновая помощь и 100-страничное руководство пользователя	200-страничное руководство пользователя
Оценка состояния	Ежеквартальное рассмотрение проекта	Ежемесячное рассмотрение управления проектом

Часть IV

Взгляд в будущее

Глава 15 ◆ **Особенности современных проектов**

Глава 16 ◆ **Экономика ПО следующего поколения**

Глава 17 ◆ **Переход к использованию современного процесса**

В части I представлены некоторые точки зрения на традиционное управление созданием ПО. В ней объективно описаны особенности традиционного проекта, экономика традиционного процесса создания ПО и принципы традиционного управления созданием ПО. Части II и III посвящены изучению схемы процесса и управленческих дисциплин, необходимых для перехода к современному методу создания ПО. Часть IV завершает представление современной схемы управления созданием ПО. Она вновь возвращается к трем описаниям результатов традиционного процесса и рассматривает современный процесс управления созданием ПО.

Материал части IV можно отнести к разряду догадок, включающих в себя некоторые ожидания (глава 15), гипотезы (глава 16) и эвристики (глава 17). В главе 15 обсуждаются ожидаемые особенности грамотно управляемого итерационного процесса разработки и описывается то, чем он будет отличаться от опыта традиционных проектов. В главе 16 представлены гипотетические наблюдения, касающиеся нового поколения экономики создания ПО. Глава 17 посвящена некоторым изменениям в культуре, необходимым для перехода к современным методам управления созданием ПО.

Глава 15

Особенности современных проектов

В главе 1 представлены пять характерных для традиционных проектов проблем. В схеме современного процесса используются некоторые подходы к решению этих проблем:

Ключевые моменты

- Особенностью современных проектов является то, что позитивные и негативные тенденции становятся более ощутимыми уже на ранних стадиях жизненного цикла.
- Интеграция начинается рано и выполняет функцию верификации рабочих продуктов.
- Многие критичные риски оказываются разрешенными к моменту завершения стадии уточнения. Стадии конструирования и ввода в действие, которым присущ основной финансовый риск, лишены всяких сюрпризов.
- Внимание по достижению основных контрольных точек сосредоточивается на демонстрируемых результатах.

1. *Затянувшаяся интеграция и позднее обнаружение ошибок, допущенных при разработке*, разрешаются посредством переноса интеграции на стадию разработки. Это достигается за счет постоянной интеграции базовой архитектуры, поддерживаемой демонстрациями выполнения основных сценариев.
2. *Позднего разрешения рисков* удается избегать за счет подхода с упреждающей разработкой архитектуры, при котором наиболее значимые элементы системы тщательно прорабатываются на ранних стадиях жизненного цикла.
3. «Паралича анализа» *функциональной декомпозиции, определяемой требованиями*, удается избежать, организуя спецификации низкого уровня в соответствии с содержанием версий, а не в соответствии с декомпозицией продукта (на подсистемы, компоненты и т.д.).

4. Противоречий между заинтересованными сторонами удается избежать за счет предоставления более осязаемых и объективных результатов на протяжении всего жизненного цикла.

5. Традиционно повышенное внимание к документации и встречам для обмена мнениями заменяется повышенным вниманием к результатам, которые можно продемонстрировать, и к четко определенным комплектам рабочих продуктов, выполненных с помощью более строгих нотаций и автоматической поддержки «безбумажной» среды.

Средства для решения этих пяти проблем, используемые современными благополучными проектами, подробно обсуждаются ниже. Способы решения четвертой и пятой проблем тесно связаны между собой и описываются в разделе 15.4. Разделы 15.5 и 15.6 посвящены анализу современных проектов в контексте моих 10 принципов управления созданием ПО и рассмотрению лучших примеров альтернативной практики.

15.1 ПОСТОЯННАЯ ИНТЕГРАЦИЯ

В результате итерационной разработки первой появляется архитектура, что допускает использование интеграции в качестве вида деятельности по верификации, осуществляемого на стадии проектирования, и позволяет обнаруживать и исправлять ошибки проектирования на ранних стадиях жизненного цикла. Такой подход исключает интеграцию типа «большого взрыва» в конце проекта за счет постоянной интеграции на протяжении всего проекта. На рис. 15.1 показаны различия между кривой прогресса благополучного современного проекта и кривой обычного традиционного проекта, представленной на рис. 1.2. Подход с упреждающей разработкой архитектуры стимулирует начало интеграции на стадии проектирования путем создания демонстрационных версий. Демонстрации не могут избавить от дефектов проектирования; напротив, они способствуют их появлению именно на стадии разработки, когда их можно эффективно исправлять в контексте целей жизненного цикла. Таким образом, удается избежать кошмара интеграции в конце процесса, позднего латания дыр и «втискивания» исправлений в ПО. В результате разработка получается более устойчивой и удобной в сопровождении.

Постоянная интеграция, присущая итерационной разработке, позволяет глубже постигнуть проблемы качества. Характеристики системы, которые во многом зависят от архитектуры (производительность, устойчивость к отказам, удобство сопровождения), становятся понятными на более ранних стадиях процесса, когда их еще можно подкорректировать, не рискуя установленными затратами и сроками. Кривая затрат, характерная для успешных проектов с итерационной разработкой, существенно отличается от опыта традиционных процессов.

Таблица 15.1 показывает отличие современного процесса с точки зрения распределения затрат по рабочим процессам. Эта таблица представляет собой комбинацию таблицы 1.1 (типичное распределение для традиционного процесса) и таблицы 10.1 (стандартное распределение для современного процесса).

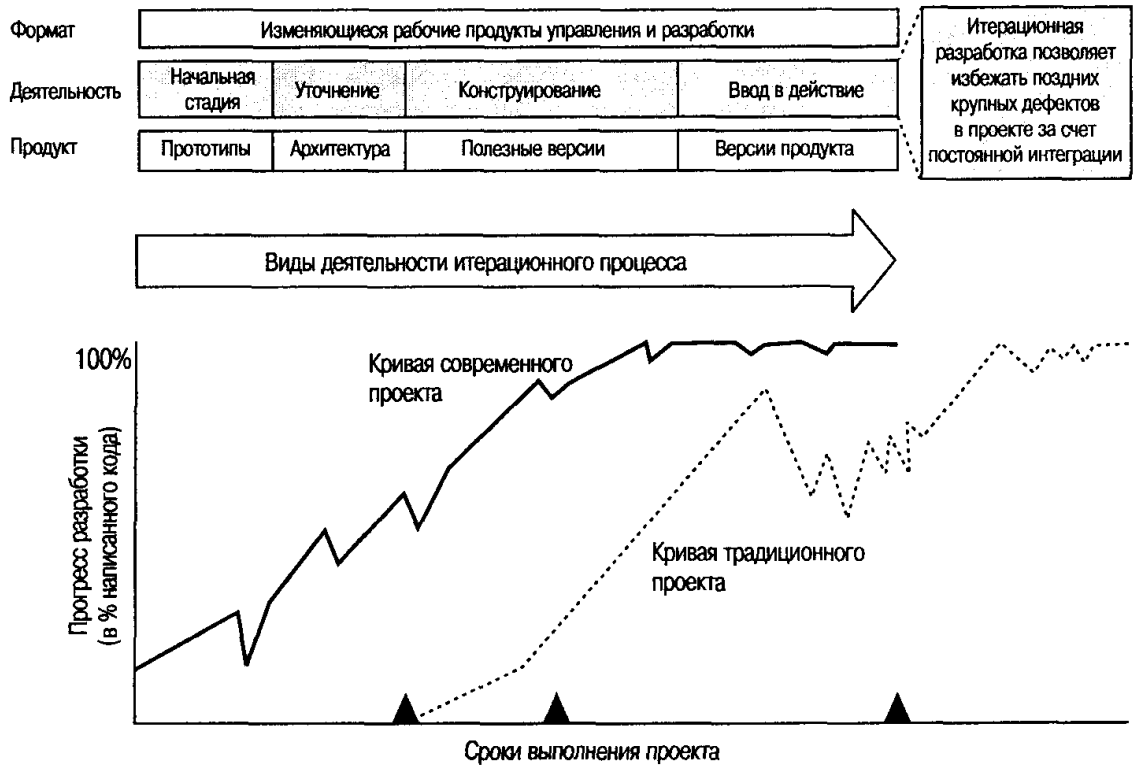


Рис. 15.1. *Вид кривой прогресса современного проекта*

Таблица 15.1.
Различия в распределении затрат по рабочим процессам между традиционным и современным подходом

Рабочие процессы	Затраты традиционного процесса	Затраты современного процесса
Управление	5%	10%
Среда	5%	10%
Требования	5%	10%
Проектирование	10%	15%
Реализация	30%	25%
Оценка	40%	25%
Внедрение	5%	5%
Итого	100%	100%

По моему опыту, главным фактором, определяющим успех современного процесса, являются затраты на оценку и тестирование на протяжении жизненного цикла. В традиционных проектах, характеризующихся неэффективной интеграцией и поздним обнаружением существенных ошибок, на интеграцию и тестирование тратится до 40% и более от суммарных ресурсов. Современные проекты, использующие зрелый итерационный процесс, позволяют получать конечный продукт, затрачивая на эти виды деятельности всего лишь около 25% от общего бюджета.

15.2 РАННЕЕ РАЗРЕШЕНИЕ РИСКОВ

На стадии разработки жизненного цикла (включающей начальную стадию и уточнение) основное внимание уделяется сопоставлению рисков и их разрешению до начала выполнения работ производственной стадии, требующих большого количества ресурсов. В традиционных проектах сначала обычно выполняются легкие задачи, в результате чего удается продемонстрировать ранний прогресс. Современный процесс начинает с самых важных 20% требований, вариантов использования, компонентов и рисков. В этом суть главного моего принципа: архитектура прежде всего. Определение архитектуры редко состоит из простых шагов, для которых легко можно достичь видимого прогресса. Эффект применения философии жизненного цикла, основанной на правилах 80/20, которые почерпнуты из опыта управления созданием ПО за последние 30 лет, дает полезную основу для управления рисками.

- **80% разработки обуславливаются 20% требований.** Постарайтесь понять ведущие требования, прежде чем направлять ресурсы на полномасштабную разработку. Не стремитесь преждевременно к высокой точности и полной трассируемости требований.
- **80% затрат на ПО связаны с 20% компонентов.** Разрабатывайте сначала критичные по затратам компоненты, чтобы планирование и контроль над затратами стали понятны уже на ранних стадиях жизненного цикла.
- **80% ошибок содержатся в 20% компонентов.** Разрабатывайте сначала критичные по надежности компоненты, чтобы деятельности по оценке хватило времени на достижение необходимого уровня зрелости.
- **80% дефектов и доработок вызываются 20% изменений.** Разрабатывайте сначала компоненты, критичные к внесению изменений, чтобы существенные изменения вносились, пока проект «не закостенел».
- **80% общего потребления ресурсов (времени выполнения, дискового пространства, памяти) приходятся на 20% компонентов.** Разрабатывайте сначала компоненты, критичные к выполнению, чтобы компромиссы по надежности, простоте внесения изменений и эффективности затрат могли быть достигнуты на как можно более ранних стадиях жизненного цикла.
- **80% прогресса обеспечиваются 20% людей.** Убедитесь, что первоначальная команда, выполняющая планирование проекта и разработку архитектуры, имеет высшую квалификацию. Адекватный план и адекватная архитектура позволят в дальнейшем достигнуть успеха со средней командой разработчиков. Неадекватный план и неадекватная архитектура, вероятно, не обеспечат успеха даже с командой, состоящей из профессионалов.

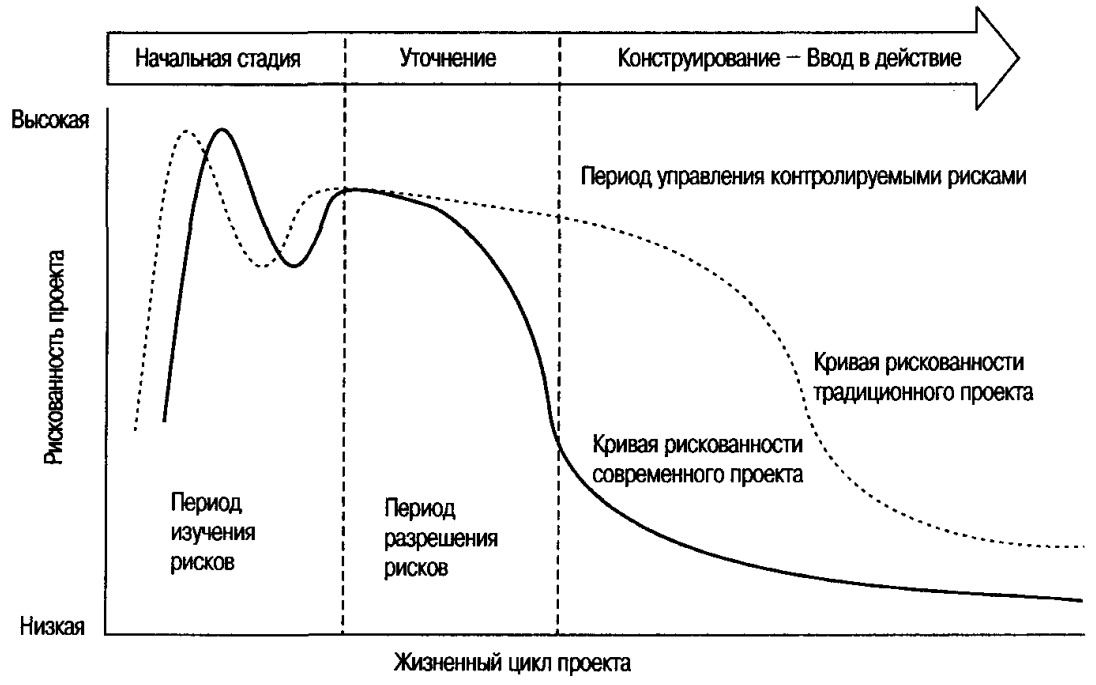


Рис. 15.2. *Кривая управления рисками типичного современного проекта на протяжении его жизненного цикла*

Рис. 15.2 позволяет сравнить кривую управления рисками для современного проекта с кривой, типичной для традиционного проекта, представленной на рис. 1.3.

15.3 ИЗМЕНЯЮЩИЕСЯ ТРЕБОВАНИЯ

Традиционные подходы разбивали требования к системе на требования к подсистемам, требования к подсистеме — на требования к компонентам, а требования к компоненту — на требования к отдельным блокам. Требования структурировались таким образом, что их трассировка оказывалась довольно простой. Притом, что на ранних стадиях жизненного цикла основное внимание уделялось в первую очередь требованиям, во вторую очередь проектированию, после чего устанавливалась полная трассировка между требованиями и элементами проекта, естественной оказывалась тенденция, при которой структура проектных решений принимала организационные формы, параллельные организационной структуре требований. Поэтому нет ничего удивительного в том, что функциональная декомпозиция проблемной области вела к функциональной декомпозиции области решения.

Большинство современных архитектур, использующих коммерческие компоненты, наследуемые компоненты, распределенные ресурсы и объектно-ориентированные методы, не может тривиально выводиться из требований, которым они удовлетворяют. На сегодняшний день между требованиями и элементами проекта устанавливаются сложные связи типа один к одному, многие к одному, один ко многим, а также условные связи, зависящие от времени и состояния.

Системные требования верхнего уровня по-прежнему представляются в форме общей концепции, однако требования более низкого уровня выражаются критериями оценки, применимыми к каждой промежуточной версии. Эти рабочие продукты (см. рис. 15.3) имеют тенденцию изменяться по ходу процесса, становясь все более и более точными по мере того, как развивается жизненный цикл и понимание требований становится все более зрелым. В этом и заключается фундаментальное отличие от подхода к работе с требованиями, при котором погоня за точностью начинается на слишком ранних стадиях жизненного цикла.

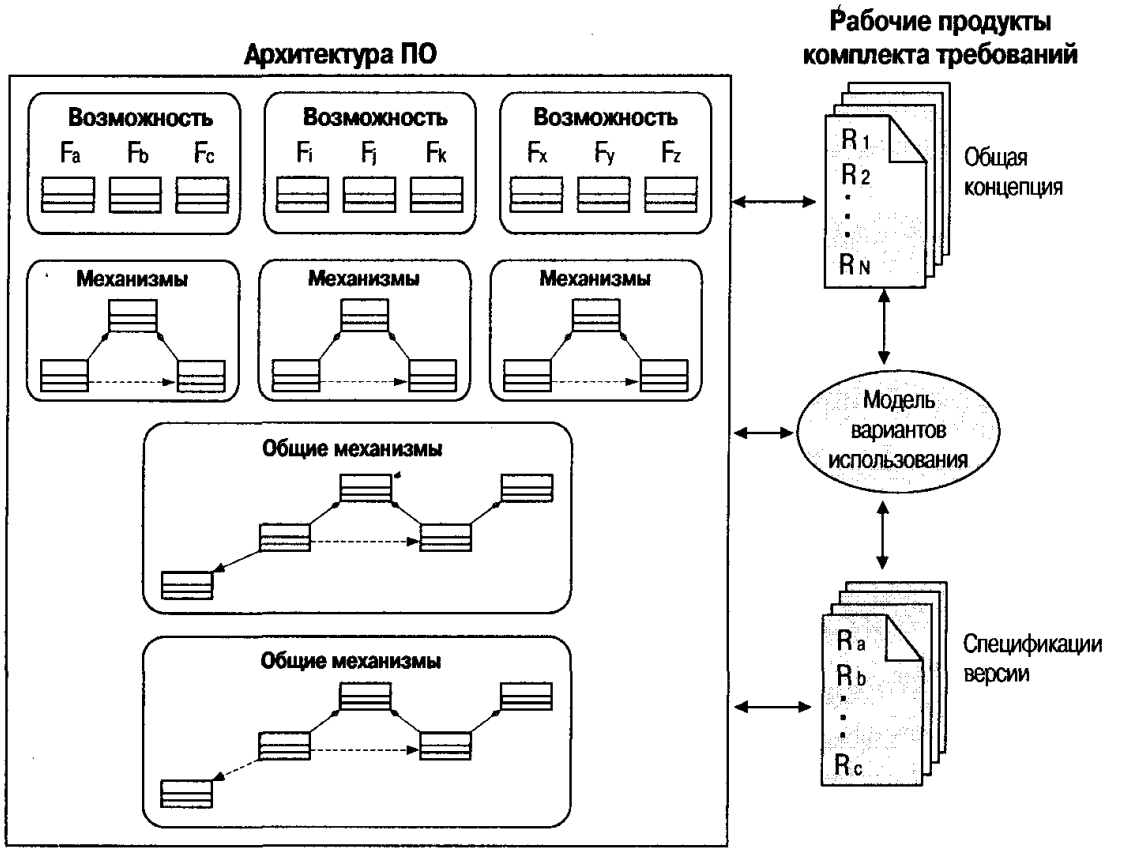


Рис. 15.3. *Организация программных компонентов, являющаяся результатом применения современного процесса*

15.4 РАБОТА РАЗЛИЧНЫХ ЗАИНТЕРЕСОВАННЫХ СТОРОН В ОДНОЙ КОМАНДЕ

Многие аспекты классического процесса разработки приводят к тому, что отношения между заинтересованными сторонами вырождаются до взаимного недоверия, это затрудняет процесс согласования требований, возможностей продукта и планов. Процесс, более близкий к итерационному, с более эффективными рабочими взаимоотношениями, позволяет достигать соглашений, основанных на объективном понимании каждого аспекта. Такой процесс требует, чтобы заказчики, пользователи и наблюдатели имели знания и опыт, касающиеся как приложений, так и ПО,

концентрировали свое внимание на получение практически полезной системы (вместо слепого стремления к соблюдению стандартов и положений контракта) и доброжелательно относились к тому, что подрядчик получит прибыль при хорошем выполнении работы. От организации-разработчика также требуется, чтобы она сосредоточила свое внимание на удовлетворении заказчика и достижении высокого качества с выгодой для себя.

Переход от обмена преимущественно бумажными документами к демонстрации промежуточных результатов является одним из важных механизмов, способствующих работе разных заинтересованных сторон в одной команде. Главные контрольные точки позволяют получать ощутимые результаты и обратную связь с пользователем. Как видно из таблицы 15.2, разработчики признаются виновными до тех пор, пока не будет доказана их невиновность: проект не будет иметь поступательного движения, пока не будут достигнуты цели демонстрации. Эта предпосылка не исключает возможности проведения повторных переговоров относительно целей по мере того, как результаты демонстрации и основной контрольной точки позволяют глубже понять природу компромиссов, присущих требованиям, проектным решениям, планам и технологии.

Таблица 15.2.
Результаты основных контрольных точек современного процесса

Видимый результат	Реальный результат
Ранние демонстрации переводят проблемы разработки и неоднозначности в осязаемую форму.	Демонстрации выявляют важные достижения и риски сложных программных систем на ранних стадиях, когда они могут быть разрешены в контексте задач жизненного цикла.
Проект не соответствует требованиям (пока).	Понимание соответствия требованиям формируется по важным аспектам (существенным для архитектуры требованиям и вариантам использования).
Проблемы в ведущих требованиях выявлены, но точное соответствие требованиям отсутствует.	Принимаются решения по изменению требований в соответствии с соглашениями по проектированию.
Проект признается «виновным, пока не доказано обратное».	Прогресс разработки и проблемы осязаемы настолько, что могут учитываться при составлении планов следующей итерации.

В таблице 15.2 очевидным результатам может сопутствовать негативное пояснение. Современный итерационный процесс, в центре внимания которого находятся демонстрируемые результаты (а не просто брифинги и бумаги), требует, чтобы все заинтересованные стороны были информированы относительно разницы между видимыми отрицательными результатами и свидетельством реального прогресса. Например, рано обнаруженный недостаток проекта, когда стоимость его исправления вполне приемлема, зачастую может рассматриваться как движение вперед, а не как большая проблема.

15.5 ДЕСЯТЬ САМЫХ ВАЖНЫХ ПРИНЦИПОВ УПРАВЛЕНИЯ СОЗДАНИЕМ ПО

Мои 10 самых важных принципов управления созданием ПО были представлены в главе 4 в качестве заднего плана для схемы процесса создания ПО и ее основополагающих принципов. С целью обобщения контуров современного проекта повторим все эти принципы вместе с описанием того, что может дать проекту успешное применение каждого принципа. По существу, предлагаемый список представляет собой краткое описание особенностей и преимуществ современного процесса так, как это видится менеджеру проекта по созданию ПО.

1. **Основывайте процесс на подходе с *упреждающей разработкой архитектуры.*** Внимание, уделяемое архитектуре на ранних стадиях, является прочным основанием для 20% работ (требования, компоненты, варианты использования, риски, ошибки), что ведет к общему успеху проекта. Добиваясь того, чтобы важные с точки зрения архитектуры компоненты были хорошо поняты и стабилизировались до детальной проработки рабочих продуктов, можно достигнуть того, что уровень дефектов и переделок на протяжении жизненного цикла будет снижаться или оставаться стабильным.

2. **Постройте *итерационный процесс всего жизненного цикла* таким образом, чтобы риски выявлялись на ранних стадиях.** Более динамичная схема планирования, базирующаяся на итерационном процессе, позволяет лучше справляться с рисками, делает производительность более предсказуемой. Первоочередное решение критичных проблем делает стадию конструирования предсказуемой и лишенной сюрпризов, а также минимально подверженной излишним затратам денег и времени.

3. **Переходите к *методам разработки, основанным на компонентах.*** Сложность работы по созданию ПО является в основном функцией количества рабочих продуктов, производимых человеком. Переходя к более компактным решениям, можно уменьшить сложность управления.

4. **Используйте *среду, позволяющую управлять изменениями.*** Динамика итерационной разработки, которая включает в себя параллельные процессы, выполняемые различными командами над одними и теми же рабочими продуктами, требует полного контроля над базовыми рабочими продуктами.

5. **Упрощайте внесение изменений за счет использования инструментов, которые поддерживают *«круговую» разработку.*** Автоматизация позволяет командам больше времени посвящать разработке и меньше времени тратить на вспомогательные действия.

6. **Содержите рабочие продукты проектирования в строгой, *основанной на моделях нотации.*** Использование инженерной нотации для проектирования способствует контролю за сложностью, объективной оценке и автоматизации анализа.

7. **Обеспечивайте процесс инструментарием для *объективного контроля за качеством* и оценки прогресса.** Показатели прогресса и качества непосредственно получаются из изменяющихся рабочих продуктов, что дает более содержательное понимание тенденций и соответствия требованиям.

8. **Для оценки промежуточных рабочих продуктов используйте *подход, основанный на демонстрациях.*** Интеграция начинается на ранних стадиях и продолжается на протяжении всего жизненного цикла. Промежуточные результаты оказываются объективными и осязаемыми.

9. **Планируйте промежуточные версии для групп сценариев использования с *переменным уровнем детализации.*** Изменяющиеся требования, разработка и планы должны находиться в равновесии. Полезные версии ПО доступны уже на ранних стадиях жизненного цикла.

10. **Используйте *конфигурируемый процесс*, который может быть экономически масштабируемым.** Методы, способы, инструменты и опыт могут непосредственно применяться в широкой области, обеспечивая повышенную отдачу от инвестиций.

На протяжении всей книги я уделял большое внимание важности баланса. Со всех точек зрения первостепенной задачей менеджера проекта по созданию ПО является нахождение верного баланса использования 10 самых главных принципов. Рис. 15.4 подводит итог теме баланса в контексте фундаментального уравнения экономики создания ПО.

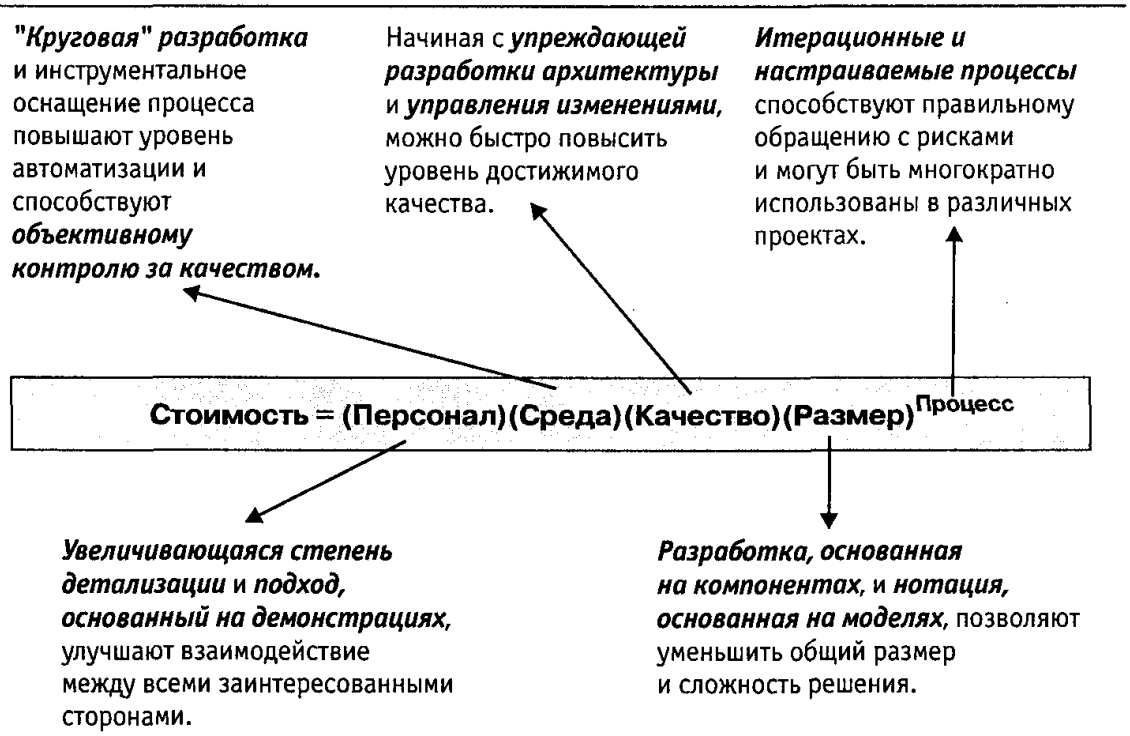


Рис. 15.4. *Взвешенное применение современных принципов для достижения высоких экономических результатов*

15.6 ЛУЧШИЕ ПРАКТИЧЕСКИЕ ПРИЕМЫ УПРАВЛЕНИЯ СОЗДАНИЕМ ПО

Многие лучшие приемы управления созданием ПО были описаны различными авторами и промышленными организациями. Одним из наиболее заметных шагов в этом направлении явилась инициатива по определению лучшей практики приобретения ПО (Sortware Acquisition Best Practices Inititaive), профинансированная Министерством обороны США с целью «усовершенствовать и реструктуризировать наш процесс управления приобретением ПО». Эта инициатива, обобщенная Брауном [Brown, 1996], поддерживается тремя органами: Совет по ПО для авиации (Airlie Software Council) (в состав которого вошли эксперты индустрии ПО), семь групп экспертов по различным проблемам (практики из правительства и промышленности) и группа менеджеров ПО (опытные менеджеры промышленных проектов). Каждый из них разработал свои рекомендации и рассмотрел работу двух других органов.

Совет по ПО для авиации обладал «необходимой структурой, в которую входили самые удачливые менеджеры крупномасштабных проектов, признанные авторы, выдающиеся консультанты и администраторы, ответственные за разработку ПО в основных компаниях». Одним из продуктов Совета стал набор из девяти лучших практических приемов. Советом была предпринята попытка сфокусировать внимание на приемах, которые могли бы оказать самый сильный эффект на совершенствование дисциплины управления ПО для крупномасштабных проектов и на контроль за сложностью в рамках таких проектов.

Эти девять лучших практических приемов приводятся ниже. Я комментирую их с точки зрения того, как они соотносятся со схемой проекта, управленческими дисциплинами и 10-ю главными принципами, рекомендованными мною. (Заимствования выделены *курсивом.*)

1. *Формальное управление рисками.*

   ▲ Использование ***итерационного процесса,*** который явно рассматривает риск, более или менее соответствует тому, о чем здесь говорится.

2. *Соглашение по интерфейсам.*

   ▲ Хотя мы можем пользоваться разными словами, здесь преследуется та же самая цель, что и в моем принципе у***прежда***ющей ***разработки архитектуры.*** Формирование основ архитектуры движет проект к достижению соглашения по различным внешним интерфейсам и самым важным внутренним интерфейсам, поскольку все они являются неотъемлемой частью архитектуры.

3. *Формальные проверки.*

   ▲ Процесс оценки на протяжении всего жизненного цикла, выполняемый параллельно с другими процессами, должен взвешенно использовать несколько различных стратегий по исправлению дефектов. Наименее важной стратегией, с точки зрения широты охвата, являются формальные проверки по причине высоких затрат человеческих ресурсов и низкого уровня обнаружения дефектов в отношении к архитектурно значимым дефектам, которые распределены по нескольким компонентам и разбросаны во времени.

4. *Планирование и управление на основе метрик.*

▲ Этот важный принцип непосредственно связан с моими принципами ***нотации, основанной на моделях***, и ***объективного контроля за качеством***. В отсутствие строгой нотации для рабочих продуктов измерение прогресса и качества вырождается до уровня приблизительных субъективных оценок.

5. *Двойной контроль качества на уровне «дюймовой гальки».*

▲ Этот прием иногда интерпретируется неправильно. Слишком многие проекты использовали точно такой подход на ранних стадиях жизненного цикла и разрабатывали чрезвычайно подробный план за очень высокую цену. Три месяца спустя, когда изменялись некоторые требования или архитектура, большую часть детально разработанного плана приходилось переделывать заново. Лучшим является подход, позволяющий поддерживать качество плана на уровне, соответствующем уровню понимания требований и архитектуры. Вместо «дюймовой гальки» я бы рекомендовал расставить контрольные точки на стадии разработки, а «дюймовую гальку» оставить для стадии производства. Это основная мысль моего принципа ***изменяющегося уровня детализации***.

6. *Сравнение прогресса всей программы с планом.*

▲ Этот прием, т.е. открытость взаимодействия между членами команды, работающей над проектом, конечно же, необходим. Ни один из моих принципов не соответствует ему напрямую. Он кажется настолько очевидным, что я оставляю его без комментариев.

7. *Отслеживание дефектов в сравнении с целевым качеством.*

▲ Этот важный принцип непосредственно связан с моими принципами ***упреждающей разработки архитектуры*** и ***объективного контроля качества***. Критичные дефекты и целевое качество определяются архитектурой. Получение рычага воздействия на эти аспекты и отслеживание тенденций их изменения являются необходимыми требованиями для достижения успеха.

8. *Управление конфигурацией.*

▲ Совет по ПО для авиации выделил управление конфигурацией как ключевое для управления сложностью и для отслеживания изменений, вносимых во все рабочие продукты. Он также признает, что автоматизация является важной из-за объема и динамики современных крупномасштабных проектов, где применение ручных методов требует высоких затрат и чревато ошибками. Такое же обоснование имеет мой принцип ***управления изменениями***.

9. *Ответственность управления за информированность людей.*

▲ Это еще один принцип управления, который кажется очевидным, и я оставляю его без комментариев.

Нетрудно заметить совпадение и общность духа между моими главными принципами и лучшими приемами Совета по ПО для авиации. Однако мне кажется, что Совет пропустил такие важные принципы, как конфигурируемость и разработка, основанная на компонентах, моделях и демонстрациях. Это кажется удивительным. Моим объяснением причин включения принципов, гласящих, что разработка должна основываться на компонентах и на моделях, является снижение сложности работ. Это та самая цель, которую провозгласил Совет по ПО для авиации. Принцип, по которому разработка должна основываться на демонстрациях, попал в число первых 10-ти прежде всего для стимуляции процесса постоянной интеграции на протяжении всего жизненного цикла и для установления более тесных связей между заинтересованными сторонами посредством более осмысленных средств взаимодействия. Поскольку Совет по ПО для авиации сосредоточился только на одной частной области — на крупномасштабных, имеющих национальное значение системах — конфигурируемость оказалась не столь существенной.

Два из приемов Совета по ПО для авиации, которые я бы не стал включать, это проверки и двойной контроль качества на уровне «дюймовой гальки». Они полезны, но на практике им уделяется чересчур много внимания, кроме того, существуют другие важные принципы, которые следовало бы учесть.

Глава 16

Экономика ПО следующего поколения

Ключевые моменты

- Экономика ПО следующего поколения должна отражать улучшение экономии при больших масштабах и повышать отдачу от вложенных ресурсов. И то и другое является реальным показателем зрелости индустрии.
- Дальнейшие технологические усовершенствования в «круговой» разработке являются базой для последующего количественного скачка в экономике ПО.
- Будущие модели для оценки стоимости должны базироваться на более совершенных элементарных составляющих, определяемых на основе хорошо понятных нотаций, таких как Unified Modeling Language.

Экономика ПО следующего поколения уже используется на практике некоторыми наиболее передовыми организациями, занимающимися созданием ПО. Многие из способов, процессов и методов, присущие описанной в этой книге схеме процесса, применяются на практике в течение нескольких лет. Однако в средней организации, занимающейся созданием ПО, еще и близко не подошли к зрелому современному процессу. В этой главе представлено несколько дерзких гипотез, касающихся будущего экономики ПО. Предлагается общая структура модели оценки стоимости, которая больше подходит для схемы рассматриваемого процесса. Мне кажется, что новый подход позволит повысить аккуратность и точность оценок стоимости ПО, что он сможет привести к впечатляющему улучшению экономии при больших масштабах ПО. Такое улучшение станет возможным благодаря совершенствованию среды разработки ПО. Наконец, я вернусь к отправным точкам выполнения традиционного проекта по созданию ПО, предложенным Боэмом, и объективно расскажу, каким образом схема процесса должна улучшать общую экономику, которую удалось достигнуть за счет организации проекта.

16.1 МОДЕЛИ СТОИМОСТИ СЛЕДУЮЩЕГО ПОКОЛЕНИЯ

Эксперты ПО придерживаются весьма разнообразных мнений относительно экономики ПО и ее проявлений в моделях оценки стоимости ПО: строки исходного кода или функциональные точки? экономия при больших масштабах или плата за большие масштабы? измерение производительности или качества? Java или C++? объектно-ориентированная или функционально ориентированная? коммерческие компоненты или изготовленные на заказ? По всем этим темам идут широкие дискуссии. Эмоциональная переоценка или недооценка — в зависимости от вашей точки зрения — не позволяет отделить факты от преувеличений. Энергичное несогласие является свидетельством того, что индустрия находится в движении, при котором быстро формируются многие конкурирующие технологии и способы. Однако одним из следствий этого является отсутствие возможности точного предсказания объема ресурсов, необходимых для конкретного проекта по созданию ПО. Точные оценки сегодня возможны, хотя честные оценки остаются неточными. Трудно улучшать эмпирические модели оценки, пока данные, касающиеся проекта и входящие в эти модели, искажены, совершенно бессвязны и базируются на различных процессах и технологиях.

Некоторые из популярных сегодня моделей оценки стоимости ПО не совсем подходят для итерационного процесса создания ПО, в котором главное внимание уделяется упреждающей разработке архитектуры. Несмотря на успехи отдельных поставщиков инструментов для оценки стоимости ПО в распространении своих экспериментальных данных, многие по-прежнему используют экспериментальную базу традиционного процесса для выполнения оценок современного проекта. Этот раздел посвящен тому, какую структуру, с моей точки зрения, следует придать модели стоимости ПО для того, чтобы она лучше всего соответствовала оценке современного процесса создания ПО. В индустрии ПО существуют модели стоимости и способы оценки, поддерживающие отдельные части данного подхода. Моя модель стоимости ПО является исключительно теоретической; у меня нет никаких эмпирических свидетельств того, что этот подход является более точным, чем современные модели стоимости. Сегодня доступен целый ряд методов и технологий, необходимых для современного процесса управления, однако не существует соответствующих завершенных проектов, которые могли бы стать объективным обоснованием моих суждений.

В модели стоимости ПО следующего поколения разработка архитектуры должна быть явно отделена от производства приложения так же, как в процессе с упреждающей разработкой архитектуры. Стоимость разработки, производства, тестирования и сопровождения базовой архитектуры является функцией масштаба, качества, технологии, процесса и опыта команды. В модели стоимости архитектуры по-прежнему будет наличествовать некоторая плата за большой масштаб (показатель степени больше единицы), поскольку ей присуща ведущая роль в таких областях, как

исследование и ориентация на разработку. После того как организации удается получить стабильную архитектуру, затраты на производство определяются экспоненциальной функцией размера, качества и сложности с более стабильными амплитудой процесса и влиянием персонала. Модель стоимости стадии производства должна отражать экономию при больших масштабах (показатель степени меньше единицы), аналогично традиционным экономическим моделям для массового производства товаров народного потребления. Гипотетическая модель стоимости для процесса с упреждающей разработкой архитектуры приведена на рис. 16.1.

С помощью моделей стоимости ПО следующего поколения нужно оценивать крупномасштабные архитектуры с экономией при больших объемах. Это предполагает, что показатель экспоненты процесса будет меньше единицы. Я объясняю это тем, что чем больше система, тем больше возможностей для применения автоматизации и для многократного использования процессов, компонентов и архитектур.

В случае традиционного процесса минимальный уровень автоматизации, поддерживающий вспомогательные виды деятельности по планированию, контролю над проектом и управлению изменениями, приводит к тому, что многие рабочие процессы становятся трудозатратными, в результате приходится платить за большой масштаб. Отсутствие автоматизации управления было характерно как для организаций, разрабатывающих множество проектов, так и для отдельных проектов. Различные виды среды и инфраструктуры следующего поколения движутся в сторону автоматизации и стандартизации многих видов управленческой деятельности, по мере роста масштаба расходуя все меньший процент от общего объема работ на вспомогательные виды деятельности.

Повторное использование общих процессов на протяжении нескольких итераций в рамках одного проекта, нескольких версий в рамках одного продукта или нескольких проектов в рамках одной организации также позволяет избавиться от платы за большой масштаб. Основных источников дефектов и доработок удается избежать за счет применения предшествующего опыта и зрелых процессов. Разработка достоверных планов на базе заслуживающих доверия норм выполнения проекта и надежных компонентов позволяет исключить другие источники дефектов и доработок. В то время как повторное использование компонентов снижает объем производственных работ, повторное использование процессов, инструментов и опыта оказывает непосредственное влияние на экономию при больших масштабах.

Другое важное отличие этой модели стоимости заключается в том, что архитектура и приложения имеют различные единицы массы (масштаб против размера) и являются представлениями области решения. Масштаб может измеряться в терминах архитектурно значимых элементов (классы, компоненты, процессы, узлы), а размер может измеряться в SLOC или в мегабайтах выполняемого кода. Эти единицы измерения отличаются от единиц измерения в проблемной области, таких как отдельные требования или варианты использования. Описание проблемной области влечет за собой определение области решений.

$$\text{Работа} = F(T_{Arch}, S_{Arch}, Q_{Arch}, P_{Arch}) + F(T_{App}, S_{App}, Q_{App}, P_{App})$$

$$\text{Время} = F(P_{Arch}, \text{Работа}_{Arch}) + F(P_{App}, \text{Работа}_{App})$$

где:

T — технологический параметр (автоматизированная поддержка среды)
S — параметр масштаба (например, варианты использования, функциональные точки, строки исходного кода)
Q — параметр качества (такой, как переносимость, надежность, производительность)
P — параметр процесса (такой, как зрелость, опыт в предметной области)

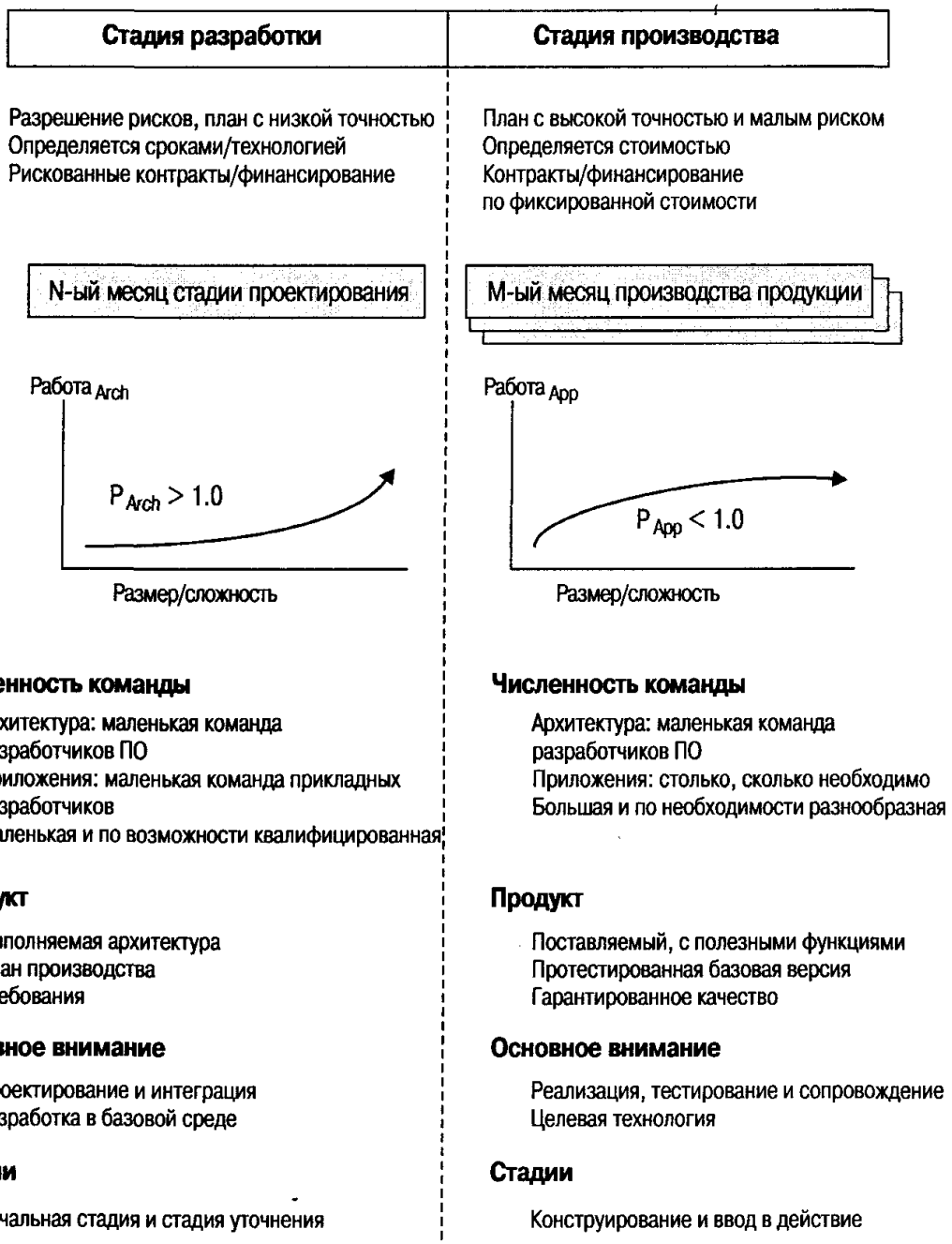


Рис. 16.1. *Модели стоимости следующего поколения*

Однако, как показано на рис. 16.2, для каждой конкретной проблемы существует множество решений, каждое из которых обладает определенной предполагаемой ценой. Ключевым аспектом при выборе из потенциальных решений является его стоимость. Из конкретных решений проблем можно получить более аккуратные и точные оценки стоимости. Из этого следует, что модель для оценки стоимости должна руководствоваться основными параметрами предполагаемого решения. Если ни одно из решений не является приемлемым по предполагаемой цене, то необходимо искать другие возможные решения либо изменять формулировку проблемы.

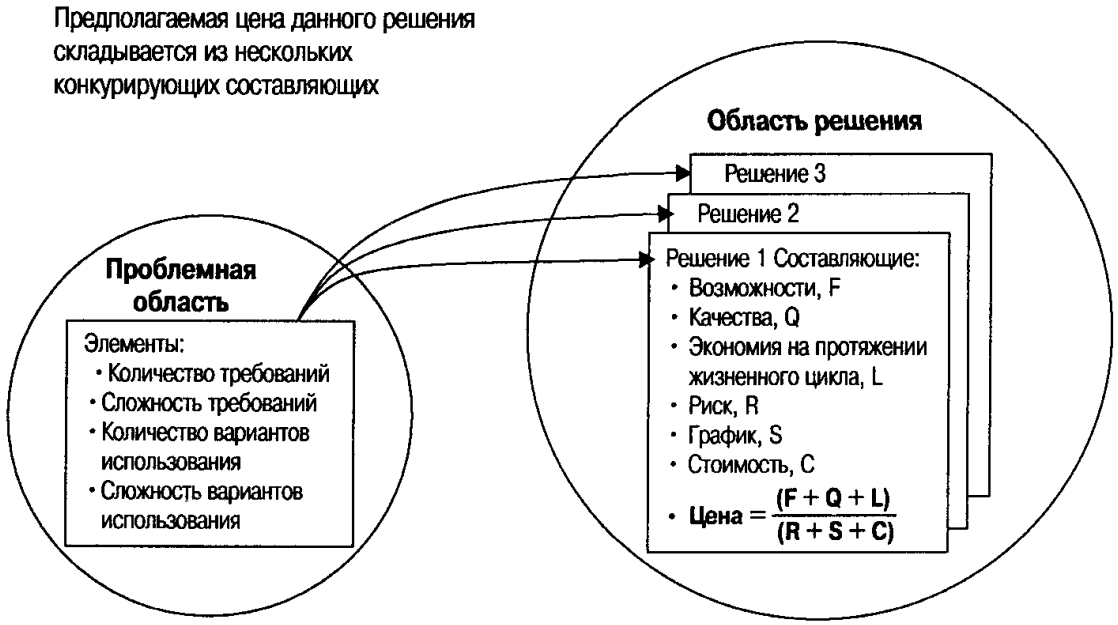


Рис. 16.2. *Дифференциация возможных решений по оценке их стоимости*

Споры между апологетами функциональных точек и апологетами строк исходного кода — хороший показатель необходимости измерения как масштаба, так и размера. Мне думается, что функциональные точки являются более точными для количественного выражения масштаба архитектуры, в то время как SLOC точнее отражают размер компонентов, из которых состоит общая реализация. Красота применения SLOC заключается в том, что сбор информации может быть легко автоматизирован, и достигнуть точности тоже не представляет труда. Однако точность SLOC как единицы измерения объема сомнительна, что может приводить к неправильной интерпретации при использовании SLOC для абсолютных сравнений между различными проектами и организациями. В частности, это справедливо для ранних фаз проекта, если SLOC применяются для измерения масштаба. Во многих проектах SLOC успешно использовались в качестве меры общего объема на поздних стадиях жизненного цикла, когда наиболее важными оказываются измерения относительного объема изменений от месяца к месяцу по мере создания выпускаемых версий.

Ценность функциональных точек заключается в том, что они лучше приспособлены для отображения общего масштаба решения, независимо от реального размера и языка программирования, выбранного для окончательной реализации. Однако функциональные точки непросто извлекать из строгих форматов представления, поэтому автоматизация и отслеживание внесенных изменений оказываются трудными или неоднозначными.

Строгая нотация для рабочих продуктов проектирования является необходимой предпосылкой каких-либо улучшений качества оценки масштаба разработки. В будущем я ожидаю появления новых возможностей для автоматизации измерения масштаба непосредственно из рабочих продуктов проектирования, представленных в виде моделей UML.

Я ожидаю двух основных улучшений в моделях оценки стоимости ПО следующего поколения:

1. Отделение стадии разработки от стадии производства приведет к необходимости отличать масштаб архитектуры от размера реализации. Это позволит получить большую аккуратность и более «честную» точность при оценках на протяжении всего жизненного цикла.

2. Строгие нотации, такие как UML, предоставят более стандартизованные единицы измерения масштаба, что предполагает возможность автоматизации и отслеживания. Такие измерения будет проще соотносить со стоимостью продукции.

Количественное выражение масштаба архитектуры ПО на стадии разработки — это область для проведения исследований. В течение следующего десятилетия в процессе создания ПО окажутся возможными два прорыва, каждый из которых будет осуществляться за счет технологических преимуществ в среде сопровождения. Первый прорыв — появление интегрированных инструментов, которые обеспечат автоматизацию преобразования информации между различными элементами требований, проекта, реализации и внедрения. Такой инструментарий приведет к более объемлющей «круговой» разработке. Основным содержанием второго прорыва будет уменьшение сегодняшних четырех комплектов фундаментальных технических рабочих продуктов до трех за счет автоматизации видов деятельности, связанных с написанием программного кода вручную, что избавит от необходимости иметь отдельный комплект реализации. Такая технологическая возможность (см. рис. 16.3) позволит получать выполняемые программы непосредственно из UML-представлений без вмешательства человека. Инструментарий визуального моделирования уже способен создавать фрагменты кода из UML-моделей, однако получение завершенных фрагментов является делом будущего.

Если первый прорыв будет рискованным, но простым, то второй станет изменением всей парадигмы. Когда команда разработчиков сможет производить рабочие продукты реализации и внедрения в рамках свободной от ошибок автоматизированной среды, процесс разработки ПО изменится разительно, примерно так же, как изменилось производство чипов при переходе к процессу «печатания».

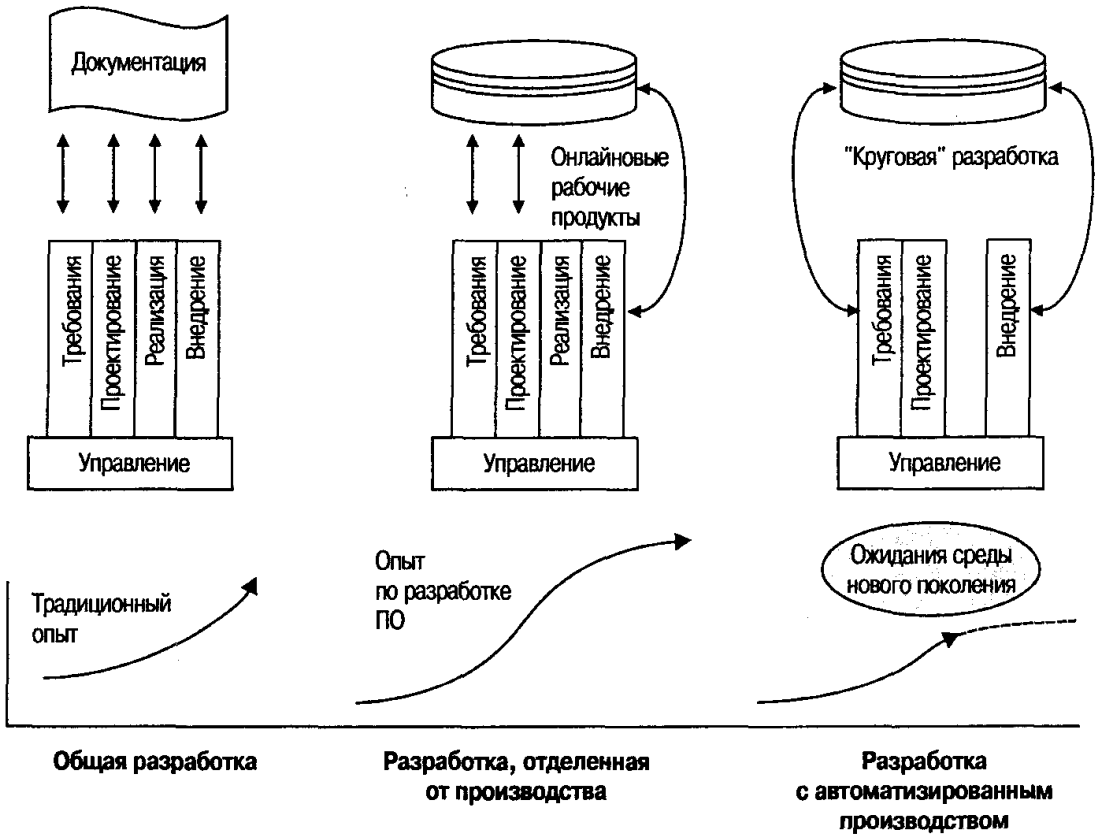


Рис. 16.3. *Автоматизация процесса конструирования в среде следующего поколения*

16.2 СОВРЕМЕННАЯ ЭКОНОМИКА ПО

В главе 1 мы познакомились с 10-ю самыми важными правилами создания ПО [Boehm, 1987] как с объективным представлением современного состояния практики по управлению созданием ПО. Эта схема может быть использована для подведения итогов некоторых важных тем в контексте экономики и для обсуждения того, каким образом должна работать современная схема управления созданием ПО. Не существует достаточного количества данных по проектам для подтверждения моих идей, но я убежден, что эти ожидаемые изменения дают хорошее объяснение того, к чему должен стремиться менеджер организации при переходе к современному процессу. (Заимствования выделены *курсивом.*)

1. *Поиск и обнаружение ошибки в ПО после его сдачи обходится в 100 раз дороже, чем поиск и обнаружение ошибки на ранних стадиях проектирования.*

   ▲ Современные процессы, компонентные технологии разработки и архитектурные схемы явно подтверждают это соотношение. Во многих областях, в том числе при решении проблем с ПО, локальных для некоторого компонента, применение инкапсуляции обеспечивает значительное снижение влияния на ресурсы, иногда на целый порядок. Тем не менее подход с упреждающей разработкой архитектуры позволяет получить преимущество в 10 – 100 раз при

исправлении ошибок в архитектуре. Соответственно, итерационный процесс дает огромный выигрыш в раннем понимании архитектуры и при выполнении видов деятельности, связанных с риском.

2. *Можно сократить срок разработки ПО на 25% от номинального, но не более.*

   ▲ Это правило должно оставаться справедливым для стадии жизненного цикла, связанной с разработкой, когда развивается интеллектуальное содержимое системы. Однако в случае, если стадия разработки оказалась успешной и привела к созданию согласованной базы — включая архитектуру, планы по созданию и требования, — величина сокращения стадии производства может быть более гибкой. Если некоторая промышленная организация применяет одну и ту же стадию разработки для различных проектов или если проектная организация использует одну и ту же стадию разработки в различных направлениях, в любом случае имеется больше возможностей для параллельной разработки.

3. *На каждый доллар, вложенный в разработку, приходится тратить два доллара на сопровождение.*

   ▲ Это правило трудно поддается обобщению, поскольку существует большое количество различных моделей сопровождения. Сравнения, производимые в абсолютных цифрах, имеют мало смысла во всех случаях, кроме сравнения однотипных проектов. Более правильным способом измерения этого показателя является определение соотношения производительности при разработке и сопровождении (см. приложение С). Одним из интересных аспектов итерационной разработки является тот факт, что граница между разработкой и сопровождением оказывается размытой. Зрелый итерационный процесс и хорошая архитектура могут существенно уменьшить объем дефектов и доработок. Мое шестое чувство подсказывает мне, что при имеющем место общем усреднении разработки и сопровождения это правило будет меняться в сторону отношения один к одному, при котором производительность процесса разработки будет не сильно отличаться от производительности процесса сопровождения.

4. *Стоимость разработки и сопровождения ПО является прежде всего функцией числа строк исходного кода.*

   ▲ Это правило гласит, что стоимость определяется прежде всего размером продукта, а фундаментальными единицами измерения размера являются строки кода. Для предыдущих поколений технологии создания ПО это было совершенно очевидно, в то время как для сегодняшних технологий, основанных на компонентах, это становится менее очевидным. Коммерческие компоненты, повторное использование и автоматические генераторы кода могут сильно исказить значение строки исходного кода. Стоимость создания по-прежнему будет определяться сложностью спецификации. Использование большего числа компонентов, типов компонентов, источников компонентов и компонентов, изготовленных на заказ, потребует большего объема работ по интеграции и приведет к повышению стоимости. Использование меньшего

числа компонентов, типов, источников и большего числа промышленных инструментов приведет к снижению стоимости. К несчастью, индустрия компонентов все еще недостаточно зрелая, чтобы отвечать таким стандартам спецификации системы, которые могли бы улучшить качество оценок стоимости. Это означает, что модели стоимости следующего поколения должны быть менее чувствительными к числу исходных строк и более чувствительными к количеству различных компонентов и к легкости их интеграции.

5. *Различия между людьми приводят к огромной разнице в продуктивности при создании ПО.*

   ▲ Для каждой организации, которая занимается разработкой и для которой интеллектуальная собственность является реальным продуктом, факторами, определяющими продуктивность, будут служить навыки персонала, умение работать в команде и мотивация. Современный итерационный процесс ограничивает, насколько это возможно, требование в привлечении высококвалифицированных сотрудников стадией разработки, когда команда еще относительно мала. На стадии производства, когда команды обычно становятся гораздо больше, зависимость от недостатка опыта и знаний снижается.

6. *Общее отношение стоимости ПО к стоимости аппаратуры продолжает расти. В 1955 г. оно составляло 15:85; в 1985 г. – 85:15.*

   ▲ Я не могу сказать точно, как это правило выглядит сегодня. Популярность персональных компьютеров и различия в стоимости ПО, в частности, в стоимости программных инструментов для персональных компьютеров, без всякого сомнения, изменили это соотношение. Главное влияние этого правила на экономику ПО заключается в том, что аппаратура продолжает дешеветь. Циклы обработки, хранение данных и пропускная способность сетей предлагают все новые возможности для автоматизации. Соответственно, все более важную роль начинает играть среда ПО. В современном процессе среда выполняет гораздо больший объем работ по учету и анализу, чем тот, что ранее выполнялся человеком. Анализ конфигурации и соответствия качества уже в значительной степени автоматизированы, очередным рубежом является автоматизация производства и тестирования.

7. *При создании ПО всего лишь около 15% работ тратится собственно на программирование.*

   ▲ За последнее десятилетие наблюдалось заметное снижение инвестиций в языки и компиляторы, за исключением языков Java и Ada 95. Инвестиции в современные технологии перекочевали в области определения зрелости процесса (например, SEI CMM), визуального моделирования (UML), автоматизированного качества ПО (автоматизация тестирования), создания компонентов (ActiveX, Java и CORBA), управления настройками, определения параметров и других аспектов разработки ПО. Объем программирования, выполняемый в рамках проекта по разработке ПО, по-прежнему остается приблизительно на уровне 15%. Различие заключается в том, что в рамках современных проектов программирование выполняется на более высоком уровне абстракции. В 1960-х гг. на один человеко-месяц в среднем

приходилось около 200 машинных команд, а в 1970-х и 1980-х гг. — около 1000. Производительность программирования в 1990-х гг. позволяет получать десятки тысяч машинных команд в месяц, хотя реально может быть произведено всего лишь несколько сотен написанных вручную строк исходного кода.

8. *Программные системы и продукты обычно стоят в три раза дороже в пересчете на одну строку исходного кода, чем отдельные программы. Продукты, состоящие из программных систем (т.е. системы систем), дороже в девять раз.*

▲ Современный процесс и современные технологии позволяют в значительной степени избавиться от платы за большой масштаб. При определенных условиях — например, в производстве отдельных, предназначенных для конкретного заказчика программных систем, обладающих единой архитектурой, единой средой и единым процессом, — удается достигнуть экономии при больших масштабах.

9. *Сквозной контроль позволяет обнаружить 60% ошибок.*

▲ Я уже обращал внимание на то, что это правило необходимо исключить из первой десятки. Проверка и контроль, выполняемые человеком, не позволяют обнаруживать критичные проблемы; они могут только помочь в их решении. Это правило нужно заменить следующим: «В то время как среда позволяет обнаружить большинство несоответствий и ошибок первого уровня, по-настоящему важные проблемы в архитектуре могут быть выявлены только посредством демонстраций и разрешены только посредством внимательного изучения человеком».

10. *80% работы выполняется 20% работающих.*

▲ Это соотношение — на все времена. Оно является основополагающей философией, которую следует использовать при планировании и при осуществлении современного процесса руководства созданием ПО.

Глава 17

Переход к использованию современного процесса

Управление созданием ПО – тяжелая работа. Технические прорывы, прорывы в организации процесса и новые инструменты облегчают ее, но дисциплина управления остается решающей для успеха проекта по созданию ПО. Новые технологические достижения будут сопровождаться появлением новых возможностей для применения программных приложений, новыми горизонтами сложности, новыми направлениями автоматизации и новыми заказчиками с различными приоритетами. Чтобы подстроиться под эти изменения, придется пересмотреть многие из укоренившихся ценностей и приоритетов управления созданием ПО. Однако установление равновесия между требованиями, проектами и планами по-прежнему будет являться основополагающей целью при управлении будущими проектами по созданию ПО так же, как это является основополагающей задачей сегодня.

Ключевые моменты

- Переход к современным процессам и технологиям создания ПО влечет за собой необходимость некоторых изменений в общей культуре, чего не всегда оказывается просто достигнуть.
- Уроки, извлеченные при переводе организаций на современный процесс, выявили несколько слагаемых успеха, которые представляют собой изменения в общей культуре по сравнению с традиционной практикой.
- Попытку осуществления значимого перехода следует предпринимать на значимом проекте. Пилотные проекты обычно не привлекают лучшие таланты, а именно наличие лучших специалистов является критичным для любого значимого перехода.

Схема управления созданием ПО, представленная в этой книге, не является революционной; бесчисленное количество проектов

использовало некоторые из ее положений годами. Однако многие из способов и дисциплин, предложенных в книге, потребуют значительного изменения парадигмы. Некоторые из изменений встретят сопротивление определенных заинтересованных сторон или определенных группировок внутри организации, выполняющей проект. Далеко не всегда легко отличить сопротивление, обусловленное общей культурой, от объективного сопротивления. В настоящей главе обобщаются наиболее важные изменения в общей культуре, к которым необходимо подготовиться с тем, чтобы снизить торможение при переходе к современному процессу.

17.1 ИЗМЕНЕНИЯ В ОБЩЕЙ КУЛЬТУРЕ

Общая культура должна претерпеть некоторые изменения для того, чтобы переход к современному процессу управления созданием ПО оказался успешным. Иногда трудно провести границу между объективным противодействием и обыкновенным упрямством. Как бы то ни было, существуют общие показатели успешного перехода к современной культуре. В этом разделе обсуждаются приблизительные индикаторы, с помощью которых удается отличить проекты, в рамках которых произошло подлинное изменение культуры, от проектов, в которых изменен лишь внешний фасад. Многие из этих индикаторов могут быть получены непосредственно из схемы процесса, описанной в предыдущих главах; другие дают эффект более низкого порядка.

- **Менеджеры низшего и среднего уровней являются исполнителями.** В организации или ее подразделениях численностью 25 и менее сотрудников не должно быть «чистых» менеджеров. Необходимость в «чистых» менеджерах возникает только тогда, когда ресурсы рабочей силы превышают этот уровень. Непосредственные навыки управления могут разниться, но компетентные менеджеры обычно проводят большую часть времени, выполняя непосредственную работу, в частности, позволяющую в первую очередь понять состояние проекта, а также действия, связанные с разработкой планов и получением оценок. Более того, человек, управляющий некоей работой, должен планировать ее. Это не означает, что менеджер должен утверждать план; это означает, что менеджер должен принимать участие в его разработке. В независимых проектах, в оценке которых я принимал непосредственное участие, хорошим индикатором проблем оказывался менеджер, который не только не являлся автором плана, но и не признавал его. Это изменение прежде всего касается менеджеров проектов по созданию ПО.

- **Требования и проектные решения изменчивы и осязаемы.** В традиционном процессе слишком много внимания уделялось составлению документов, описывающих программный продукт, и слишком мало — осязаемым шагам по непосредственному созданию продукта. Основные контрольные точки описывались исключительно в терминах создания определенных документов. Организации-разработчики больших проектов, выполняемых по контракту, были

вынуждены изводить тонны бумаги для того, чтобы продемонстрировать соответствие очередной контрольной точке и получить очередную порцию финансирования, вместо того чтобы направлять свою энергию на задачи, решение которых позволило бы уменьшить риск и создать качественное ПО. Итерационный процесс требует реального построения последовательности все более завершенных систем, которые позволяют продемонстрировать архитектуру, делают возможными объективные переговоры по требованиям, подтверждают правильность технического подхода и указывают пути разрешения ключевых рисков. В идеальном варианте все заинтересованные стороны должны сосредоточиваться на этих «реальных» контрольных точках с постоянным приращением полезной функциональности, а не на спекулятивных описаниях на бумаге того, каким образом все это должно выглядеть в окончательном варианте. Переход к среде, менее зависимой от документации, будет встречен командами разработчиков с распростертыми объятиями; скорее всего, сопротивления следует ожидать со стороны обычных наблюдателей за выполнением контрактов.

- **Приветствуются честолюбивые демонстрации.** Целью демонстраций на ранних стадиях жизненного цикла является выявление пороков разработки, а не создание видимости. На ранних этапах заинтересованные стороны не должны принимать слишком близко к сердцу ошибки, отклонения или несовершенные решения. Для версий, запланированных на ранних стадиях, критерии оценки являются целями, а не требованиями. Если встреченным трудностям будет придаваться чрезмерное значение, организации-разработчику придется планировать последующие итерации менее честолюбивыми. С другой стороны, заинтересованные стороны вряд ли потерпят не до конца разрешенные проблемы. Если не взяться во время за негативные тенденции, они могут привести к необходимости возвращаться назад для внесения серьезных изменений. Для решения проблем необходимо открыто и внимательно доводить их до конца. Управляющая команда, вероятнее всего, будет сопротивляться этому переходу (особенно в случае, если проект был переоценен), поскольку он будет выставлять напоказ каждую связанную с разработкой или с процессом проблему, а при использовании традиционного процесса их легко скрыть. По этой же причине заказчики, пользователи и команда разработчиков приветствуют такой переход.

- **Плохое или хорошее выполнение проекта на ранних стадиях более очевидно.** При итерационной разработке успех порождает успех, а ранние неудачи чреваты поворотом разработки вспять. Опыт реальных проектов еще раз подтверждает, что именно ранние стадии проекта определяют его успех или неудачу. Следовательно, задача первостепенной важности — гарантировать, что стадии планирования и создания архитектуры осуществляются самой лучшей из возможных команд. Если эти стадии выполнены хорошими командами и выполнены успешно, то проект может быть успешно

завершен средними командами, постепенно превращающими приложения в конечный продукт. Если же планирование и разработка архитектуры проведены ненадлежащим образом, то даже самые лучшие в мире специалисты по программированию и тестированию, вероятнее всего, не сумеют достигнуть успеха. Казалось бы, никто не должен сопротивляться найму хорошей команды с самого начала. Однако большинство организаций имеет весьма ограниченные ресурсы для такого рода должностей и неуверенно себя чувствует при выполнении необходимых кадровых назначений.

- **Первые шаги будут несовершенными.** Внешние заинтересованные стороны, такие как заказчики и пользователи, не должны ожидать, что первые варианты будут работать в соответствии со спецификациями, будут завершенными, будут абсолютно надежными или будут иметь те качество и быстродействие, которые требуются от конечного продукта. С другой стороны, от организации-разработчика должны требоваться — и она обязана это демонстрировать — осязаемые улучшения на каждом последующем шаге. Такая тенденция свидетельствует о движении по направлению к спецификациям. Объективное количественное определение изменений, проблем и усовершенствований будет показывать качество процесса и среды с точки зрения выполнения будущих работ. Заказчики и пользователи с трудом воспринимают пороки первых версий, хотя на них и производят впечатление дальнейшие улучшения. Менеджеры и команда разработчиков воспринимают несовершенство как естественную часть процесса.

- **Рабочие продукты не так важны вначале, они гораздо важнее позже.** Беспокоиться о деталях (сопоставимости, тщательности и завершенности) комплектов рабочих продуктов до получения основ, которые оказываются пригодными и достаточно стабильными для того, чтобы оправдать затраты времени на анализ этих факторов качества, означает тратить время впустую. Начальные циклы разработки проекта и драгоценные ресурсы будут растранжирены на увеличение количества и качества рабочих продуктов, которые могут быстро оказаться устаревшими. Команда разработчиков воспримет этот переход со всей душой, а вот обычные наблюдатели за выполнением контрактов будут сопротивляться недостатку внимания, уделяемому завершенности на ранних стадиях.

- **Реальные проблемы выявляются и решаются систематически.** В успешных проектах признается, что требования и разработка изменяются вместе, с непрекращающимися переговорами, заключением соглашений и взаимными уступками, направленными на достижение наилучшего качества, вместо того чтобы слепо цепляться за двусмысленные положения контракта. В рамках благополучного, успешно развивающегося проекта несложно провести границу между реальными и кажущимися трудностями. В зависимости от ситуации такое изменение в общей культуре может затронуть практически любую команду.

- **Обеспечение качества — забота каждого, а не отдельная дисциплина.** Во многих организациях существует группа обеспечения качества. Я принципиально против концепции отдельных видов работ, команд и рабочих продуктов по обеспечению качества. Обеспечение качества должно органически вплетаться в каждую должность, в каждый вид деятельности, во все материалы. Реальное обеспечение качества измеряется реальным прогрессом и объективными данными, а не списками ошибок, встречами и проверками, выполняемыми вручную. Менеджер проекта по созданию ПО или исполняющий его обязанности должен взять на себя роль гаранта того, что обеспечение качества является неотъемлемой частью процесса. Традиционный контроль, осуществляемый независимой командой проверяющих, заменяется самоконтролем командной работы в организации, которой присущи зрелый процесс, общие цели и общие стимулы. Обычные менеджеры и персонал, занимающийся обеспечением качества, будут сопротивляться такому переходу. Команда разработчиков примет его с удовольствием.

- **Проблемы, связанные с выполнением проекта, обнаруживаются на ранних стадиях жизненного цикла.** Проблемы, связанные с выполнением ранних стадий проекта, всплывали на поверхность почти в каждом успешном проекте из тех, что я знаю. Эти проблемы являются признаком несовершенной разработки, но одновременно и признаком совершенного процесса разработки. Обычно проблемы выполнения проекта касаются всех заинтересованных сторон. Разработчики с радостью принимают акцент на демонстрациях на ранних стадиях и возможность оценить и проверить достигнутые соглашения в последующих версиях.

- **Инвестиции в автоматизацию являются необходимыми.** Поскольку проекты с итерационной разработкой требуют широкой автоматизации, важно избегать недостаточного инвестирования в основную среду. Не менее важно для всех заинтересованных сторон заполучить интегрированную среду, обеспечивающую эффективное участие в итерационной разработке. В противном случае все взаимодействие с организацией-разработчиком сведется к обмену бумагами и будет сопровождаться большинством проблем, присущих традиционному процессу. Против таких инвестиций могут выступать менеджеры организации, уделяющие слишком много внимания краткосрочным финансовым результатам, либо персонал проекта, который отдает предпочтение отдельному проекту перед глобальным решением, служащим и проекту, и целям организации.

- **Хорошая организация, разрабатывающая ПО, должна приносить больше прибыли.** В области создания коммерческого ПО это не является проблемой. В случаях же создания ПО по контракту, особенно в правительственных контрактах, это представляет немалую проблему. Составной частью противоречивого характера процесса приобретения ПО и заключения контракта является

внимание, уделяемое тому, чтобы прибыль исполнителя оказалась в приемлемых границах (обычно от 5% до 15%). Хорошая работа исполнителя, правильная оценка разработки или применение принципа повторного использования могут привести к тому, что потенциальные границы уровня прибыли исполнителя будут превышены. Как только заказчикам (либо пользователям или наблюдателям) становится известно об этой тенденции, с неизбежностью начинает оказываться давление с целью использования этих «лишних» ресурсов для внесения изменений, выходящих за рамки контракта, до тех пор пока величина прибыли не вернется в исходные границы.

Прямым следствием этого является то, что прибыль как движущая сила, лежащая в основе коммерческих сделок и стимулирующая эффективность, замещается сложными контрактными стимулами (и конфликтами производителя с заказчиком), что обычно менее оптимально. Часто это приводит к тому, что исполнители не видят экономического стимула для снижения затрат, и, естественно, у них нет практически никакого стимула идти на риск, который может принести большую отдачу. С другой стороны, исполнители могут с легкостью «съесть» огромные средства (прибыль при этом обычно остается на небольшом уровне), не имея никаких производственных результатов и не неся никакой ответственности за плохую работу.

Во имя процветания индустрии ПО хорошие исполнители должны поощряться (получать большую прибыль), а плохие исполнители наказываться (получать меньшую прибыль). Заказчик, который приобретает хороший продукт за разумную цену, должен быть счастлив, если исполнитель имеет хорошую прибыль. Позволять плохо работающим исполнителям работать плохо и дальше не является благом ни для кого. Это одна из областей, где коммерческая сфера оказывается более эффективной, чем сфера правительственных контрактов.

17.2 ПОДВЕДЕНИЕ ИТОГОВ

Итак, традиционный процесс создания ПО имеет следующие характеристики:

- Последовательный переход от требований к разработке, кодированию и тестированию.
- Достижение 100%-ной готовности всех видов рабочих продуктов на каждой стадии жизненного цикла.
- Рассмотрение всех требований, рабочих продуктов, компонентов и т.д. на равных основаниях.
- Достижение высокоточного соответствия между всеми рабочими продуктами на каждой стадии жизненного цикла.

К основным характеристикам современного процесса итерационной разработки относятся:

- Непрерывная «круговая» разработка, начиная от требований и заканчивая тестированием, на постоянно меняющемся уровне абстракции.
- Максимально полное понимание определяющих требований (20%) настолько рано, насколько это практически осуществимо.
- Совершенствование рабочих продуктов вглубь и вширь, основанное на приоритетах, определяемых управлением рисками.
- Отнесение анализа на предмет завершенности и непротиворечивости на более поздние стадии жизненного цикла.

Схема современного процесса направлена против главных причин платы за большой масштаб, присущей традиционному процессу создания ПО. На рис. 17.1 показано следующее поколение процесса выполнения проекта по созданию ПО с помощью графика зависимости прогресса разработки от времени, причем прогресс определяется как процент написанного кода (который можно продемонстрировать в его окончательной форме). (Рисунок имеет тот же формат, что и рис. 1.2 и 15.1.)

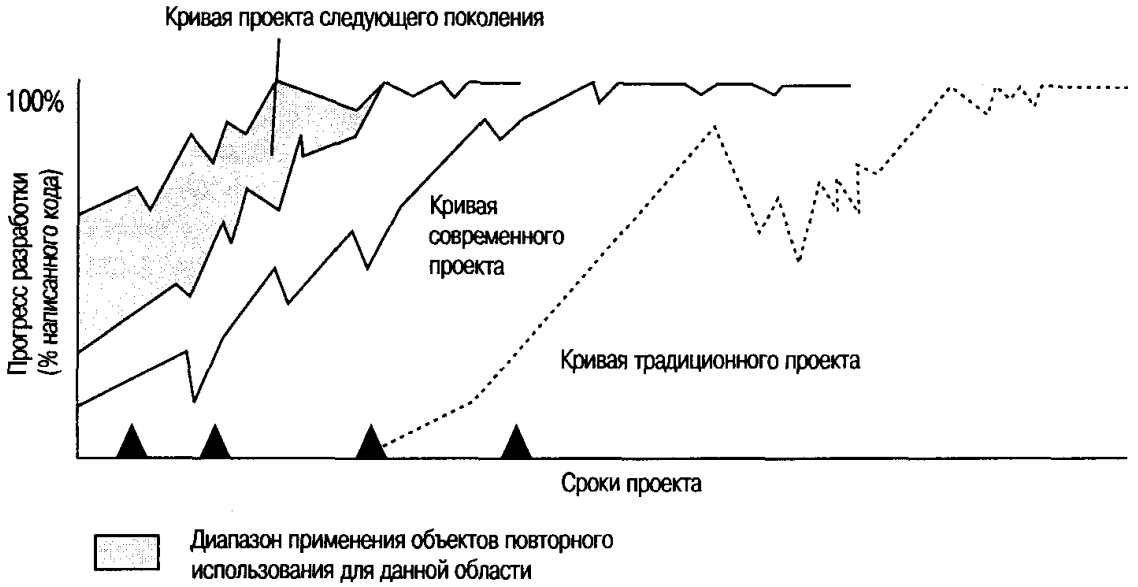


Рис. 17.1. *Процесс выполнения проекта следующего поколения*

Моей целью было объяснить, каким образом можно переместиться в верхнюю, темную область, используя современный процесс, который поддерживается совершенной, полностью интегрированной средой и архитектурой, основанной на компонентах. Преуспевающие организации смогут поставлять программные продукты, которые большей частью конструируются из уже существующих компонентов за меньшее на 50% время, с меньшими на 50% затратами ресурсов на разработку и которые сопровождаются командами на 50% меньшими тех, что требуются для современных систем.

Переход любой организации к новым способам и технологиям всегда сопровождается мрачными предчувствиями и опасениями относительно возможной неудачи. Сохранение существующего статус-кво и доверие к уже испробованным методам обычно признается наиболее безопасным путем. В индустрии ПО, в которой для большинства организаций процент успешно завершенных проектов весьма мал, путь сохранения статус-кво не всегда является самым безопасным. Когда организация решается на осуществление перехода, то поборниками перехода внутри организации и деятелями со стороны обычно даются две рекомендации, основанные на традиционном подходе: (1) опробовать новые технологии на небольших пилотных программах и (2) быть готовыми к дополнительным затратам ресурсов – денег и времени – на первый проект, на котором осуществляется переход. Я считаю обе рекомендации непродуктивными.

Небольшие пилотные программы редко приводят к важным изменениям парадигмы. Опробование нового небольшого способа, инструмента или метода на скоротечной маломасштабной работе – скажем, менее трех месяцев и всего лишь несколько исполнителей – зачастую дает хорошие результаты, начальный толчок или подтверждение концепции. Проблема с пилотными программами заключается в том, что они почти никогда не находятся на критичных путях развития организации. Соответственно они не заслуживают привлечения специалистов высшей категории, адекватных ресурсов и внимания менеджеров.

Наиболее плодотворные изменения в организационной парадигме, которые мне когда-либо встречались, являлись результатом приблизительно такого набора обстоятельств: организация выбирала наиболее критичный для нее проект и персонал высшего разряда, предоставляла адекватные ресурсы и требовала получения высоких результатов. С другой стороны, если организация ожидает от нового метода, инструмента или технологии неблагоприятного влияния на результаты выполнения новаторского проекта, такие ожидания обычно оправдываются. Почему? Потому что ни один менеджер организации не будет сознательно оказывать неблагоприятное воздействие на наиболее важные проекты организации, и такие проекты – это место, где собираются лучшие люди. Следовательно, новаторский проект окажется не критичным проектом, над которым работают не критичные сотрудники, от которых не приходится ожидать многого. Такие непритязательные ожидания зачастую оказываются самосбывающимся пророчеством.

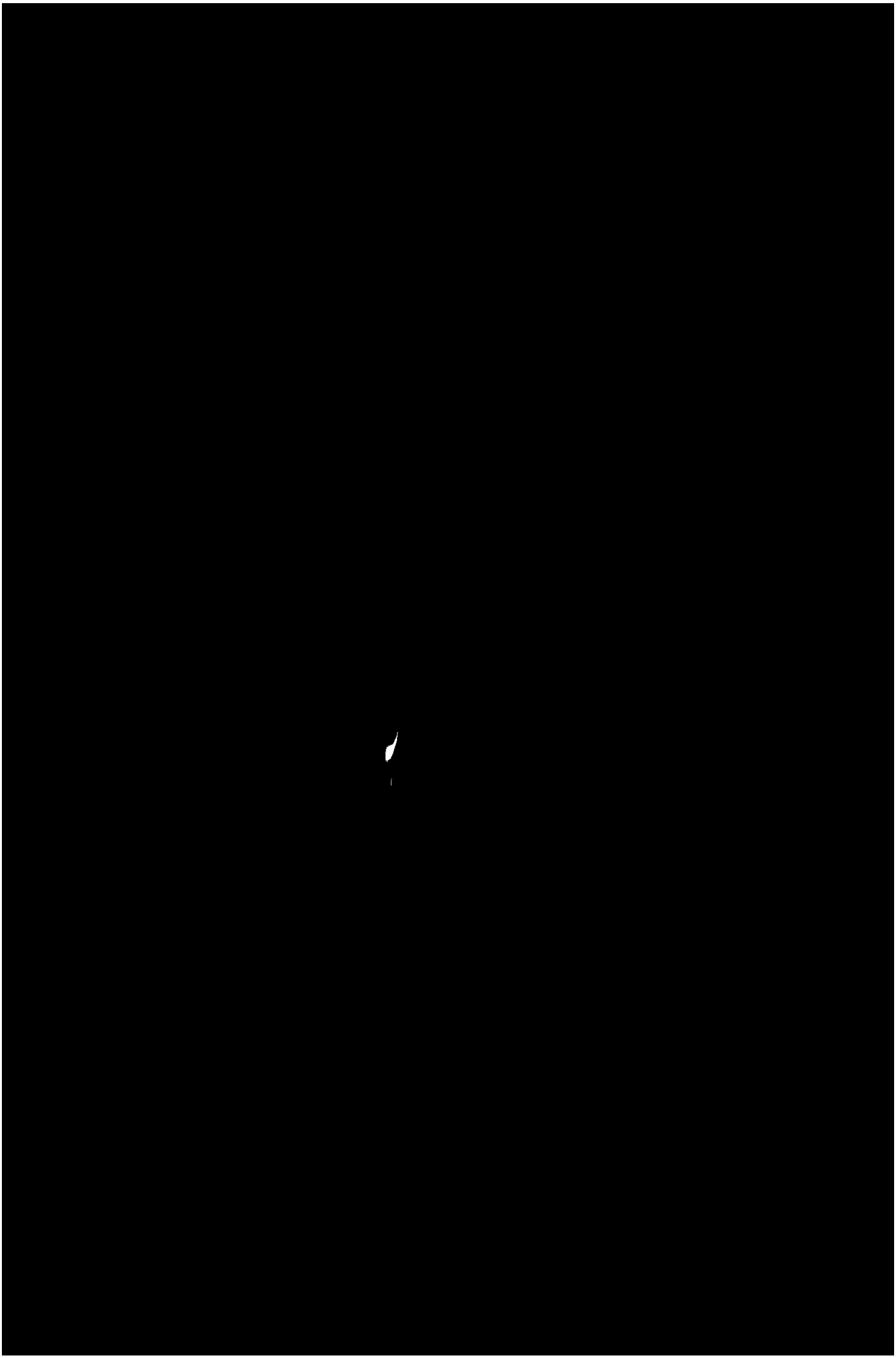
Лучший способ перейти к более зрелому итерационному процессу разработки, который поддерживает технологии автоматизации и современную архитектуру, это произвести выстрел:

- *Готовься.* Выполните свое домашнее задание. Проанализируйте современные подходы и технологии. Определите (или усовершенствуйте, или оптимизируйте) ваш процесс. Обеспечьте его совершенной средой, инструментарием и компонентами. Тщательно спланируйте.

- *Целься.* Выберите критичный проект. Наберите хорошую команду, обеспечьте ее дополнительными ресурсами и потребуйте высоких результатов.
- *Огонь.* Выполните организационный план и план проекта строго и до конца.

Часть V

Практические примеры и вспомогательные материалы



Приложение A ◆ **Состояние дел в области управления созданием ПО**

Приложение B ◆ **Оценочная модель стоимости COCOMO**

Приложение C ◆ **Метрики изменений**

Приложение D ◆ **Практический пример: CCPDS-R**

Приложение E ◆ **Усовершенствование процесса и его соответствие CMM**

Обоснование материалов, представленных в книге, охватывает бесчисленные работы множества людей над многими проектами в различных организациях. Большая часть моих соображений и рекомендаций базируется на уроках, извлеченных из реальных проектов. В эту часть включены приложения, в которых содержатся некоторые взгляды на историю и подробные объяснения, касающиеся практического использования.

- Приложение A обобщает три различные точки зрения на состояние дел в области управления созданием ПО в середине 1990-х гг.
- Приложение B рассказывает об эволюции модели COCOMO от самого начала до текущей версии — COCOMO II. Эта тема является контекстом для моих взглядов на эволюцию экономики ПО.
- Приложение C содержит дополнительные сведения о происхождении и обосновании метрик изменений, рекомендованных в главе 13.
- Приложение D предлагает подробный практический пример успешного крупномасштабного проекта по созданию ПО. Он является богатым источником реальных примеров, которые могут использоваться в качестве точек отсчета для других проектов. При изучении практического примера того или иного проекта рассматриваются темы, которые интересны лишь узкому кругу людей, однако результаты отдельных тем могут оказаться интересными для всех. Это касается общей культуры, подхода и результатов проекта, которые и определяют модель хорошо управляемого зрелого процесса создания ПО.
- Приложение E оценивает процесс, представленный в этой книге, с помощью «Вопросника для определения зрелости» (Maturity Questionnaire), разработанного SEI. Он позволяет получить приблизительное представление о зрелости, непротиворечивости и завершенности процесса относительно общепринятых точек отсчета для оценки процессов.

Приложение A

Состояние дел в области управления созданием ПО

Три проведенных в середине 1990-х гг. важных анализа дали похожие результаты относительно состояния индустрии разработки ПО. Во всех было получено заключение, что процент успеха проектов по созданию ПО чрезвычайно низок. Данное приложение обобщает результаты этих анализов.

Ключевые моменты

- ▲ В индустрии ПО практика управления его созданием по-прежнему соответствует незрелому процессу, характеризующемуся излишними объемами дефектов и доработок.
- ▲ Около 10% традиционных проектов завершаются успешно, при этом успех определяется как соответствие ожиданиям заказчика по стоимости, срокам, качеству, набору возможностей и получению прибыли.
- ▲ Факторы управления созданием ПО являются определяющими для успеха или неудачи проекта.

Примеры успешных и неудачных программных систем

Книга «Patterns of Software Sysems Failure and Success» [Jones, 1996]) является подробным анализом состояния индустрии ПО. Джонс проанализировал тысячи проектов, сгруппировав их по шести направлениям: системное ПО, информационные системы, коммерческое ПО, ПО для управления ресурсами, ПО для военных нужд и ПО для конечных пользователей. Общие оценки коренных причин удач и неудач проектов по созданию ПО, выполненные автором, сведены в таблицу A.1.

Таблица А.1.
Технологии, используемые при осуществлении проектов по созданию ПО

Технологии неудачных проектов	Технологии удачных проектов
Отсутствие «исторических» данных о ходе выполнения проекта*	Точное измерение параметров проекта*
Неудача при попытке использования инструментов для автоматизации оценок*	Раннее использование инструментов для выполнения оценок*
Неудача при попытке использования инструментов для автоматизации планирования*	Использование инструментов планирования на протяжении всего проекта*
Неудача при попытках отследить прогресс или пройденные контрольные точки*	Формализованные отчеты об успехах*
Неудача при попытке использования эффективной архитектуры*	Формализованное планирование архитектуры*
Неудача при попытке использования эффективных методов разработки*	Формализованные методы разработки*
Неудача при попытке рассмотрения разработки	Формализованное рассмотрение разработки
Неудача при попытке использования проверок кода	Формализованные проверки кода
Неудача при попытке использования формализованного управления рисками*	Формализованное управление рисками*
Неформализованное, неадекватное тестирование	Формализованные методы тестирования
Определение спецификаций и проектирование, выполнявшиеся вручную	Автоматизированное определение спецификаций и проектирование
Неудача при попытке использования формализованного управления конфигурацией*	Автоматизированное управление конфигурацией*
Изменение более 30% требований пользователя*	Изменение менее 10% требований пользователя*
Ненадлежащее использование языков 4GL	Использование подходящих языков
Излишняя и неопределенная сложность	Контролируемая и определяемая сложность
Недостаточное повторное использование сертифицированных материалов или его полное отсутствие	Значительное повторное использование сертифицированных материалов
Неудача при попытке описания элементов баз данных	Формализованное планирование баз данных

* Факторы, влияющие на управление проектом

Джонс приводит интересное наблюдение, касающееся этой таблицы:

Представляется одновременно интересным и значимым, что первые шесть из шестнадцати [так в оригинале] *технологических факторов, определяющих провалы, являются конкретными неудачами в области управления проектами, кроме того, три технологических недостатка из всех остальных могут быть так или иначе отнесены на счет неправильной практики руководства.*

Джонс определяет также культурные и человеческие факторы, которые отличают успешные проекты от неудачных. Они представлены в таблице А.2.

Таблица А.2.
Социальные факторы, наблюдаемые в проектах по созданию ПО

Неудачные проекты	Удачные проекты
Излишнее давление сроков	Реалистичные ожидания по срокам
Административный отказ от оценок	Административное приятие оценок
Серьезные трения с клиентами	Сотрудничество с клиентами
Разногласия по вопросам общей политики	Совпадающие управленческие цели
Недостаточное внутрикомандное взаимодействие	Слаженное внутрикомандное взаимодействие
Неопытные старшие администраторы	Опытные старшие администраторы
Недостаток практики у менеджеров проекта	Квалифицированный менеджмент проекта
Неквалифицированный технический персонал	Квалифицированный технический персонал
Использование неспециалистов для выполнения критичных работ: обеспечения качества, тестирования, планирования, оценок	Использование специалистов для выполнения критичных работ: обеспечения качества, тестирования, планирования, оценок

Примеры успехов и неудач оцениваются с различных точек зрения. Джонс детально описывает отличия между шестью направлениями и между проектами разного масштаба. Одной из главных его посылок является общность этих факторов для всех областей.

Я могу согласиться с большинством общих посылок, обобщенных в этих двух таблицах, но у меня несколько иное мнение по вопросу об относительной важности различных факторов и деталей реализации, связанных с успешным применением некоторых технологий. Например, три самых первых фактора в таблице А.1, может быть, и являются наиболее распространенными характеристиками, но я не думаю, чтобы именно они были наиболее важными определяющими факторами успехов и неудач. Мои взгляды на этот вопрос изложены в главе 4.

«Хаос»

В отчете «Chaos» [Standish Group, 1995] основное внимание уделяется индустрии коммерческого ПО и даются три основных вывода:

- В 1995 г. американские компании потратят на закрытые из-за неудач проекты $81 млрд.
- 31% проектов от общего числа изученных оказались закрыты до своего завершения.
- 53% проекта по созданию ПО превысили свой первоначальный бюджет более чем на 50%.

- В больших компаниях только 9% проектов по созданию ПО были выполнены в срок и уложились в рамки бюджета. Для средних и малых компаний аналогичные значения возрастают до 16% и 28% соответственно.

Отчет определяет десять главных причин, обеспечивающих проектам удачу, и десять главных причин, по которым они оказываются рискованными. (Рискованные проекты называются в нем «проблемными».) Эти факторы сведены в таблицу А.3. Большая часть отчета «Хаос» касается проблем и препятствий в том виде, как они воспринимаются менеджерами корпоративных информационных систем. Рассмотрению возможных решений в отчете уделяется минимальное внимание. Тем не менее в нем содержится описание лекарства, необходимого для лечения болезни; этот подход в большей степени ориентирован на процесс, а не на борьбу с отдельными симптомами.

Таблица А.3.
Факторы, влияющие на успех проектов по созданию ПО

Успешные проекты	% ответов	Проблемные проекты	% ответов
Вовлечение пользователей	15.9	Неучастие пользователей	12.8
Поддержка высшего руководства	13.9	Неполные требования	12.3
Ясное изложение требований	13.0	Изменяющиеся требования	11.8
Надлежащее планирование	9.6	Отсутствие поддержки руководства	7.5
Реалистичность ожиданий	8.2	Технологическая некомпетентность	7.0
Более мелкие контрольные точки проекта	7.7	Недостаток ресурсов	6.4
Компетентность персонала	7.2	Нереальные ожидания	5.9
Право собственности на продукт	5.3	Неясные цели	5.3
Ясность общей концепции и целей	2.9	Нереалистичные временные рамки	4.3
Тщательно выполняющие свои обязанности, внимательные работники	2.4	Новизна технологии	3.7
Другое	13.9	Другое	23.0

В отчете утверждается следующее:

Исследование, выполненное Standish Group, также показало, что более жесткие временные рамки с предоставлением пользователю программных компонентов как можно раньше и чаще способствуют увеличению вероятности успеха. Более жесткие временные рамки приводят к итерационному процессу разработки, созданию прототипов, реализации, тестированию и внедрению небольших элементов. Такой процесс известен под названием «растущее ПО»

в отличие от старой концепции «разрабатываемого ПО». Растущее ПО позволяет привлекать пользователя на более ранних стадиях, у каждого компонента есть автор или небольшая группа авторов, а ожидания реалистичны. Кроме того, каждый программный компонент имеет ясное и точное описание и набор целей. Компоненты ПО и небольшие проекты обычно оказываются менее сложными. Упрощение проекта – предприятие, имеющее смысл, поскольку сложность приводит только к путанице и возрастанию стоимости.

В отчете «Хаос» отражены убеждения, распространенные среди менеджеров ПО, в частности то, что причины успеха или неудачи кроются прежде всего в процессе определения требований. Из приведенных данных следует: если организация осознает, что именно она создает (требования), то вопрос о том, как это должно строиться (процесс), не представляет большой проблемы. На самом деле это далеко не так: на определение требований обычно уходит около 10% ресурсов жизненного цикла; остальные 90% тоже должны быть задействованы надлежащим образом. Поскольку работа по определению требований преобладает на ранних стадиях жизненного цикла, удобно использовать требования в качестве козла отпущения. В отличие от того, что вытекает из приведенных данных, рекомендации, выдаваемые в отчете о разбиении проблемы на небольшие части, являются проницательными и не противоречат духу современного итерационного процесса.

Доклад рабочей группы Научного совета по обороне о приобретении ПО для оборонных целей на коммерческой основе

Отчет Научного совета по обороне «Report of the Defense Science Board Task Force on Acquiring Defense Software Commercially» [Defence Science Board, 1994] содержит следующие заключения:

- Современная практика Министерства обороны несовместима с коммерческой практикой.
- Подходы Министерства обороны к управлению созданием программ не способствуют использованию коммерческой практики.
- Наблюдается дефицит подготовленного соответствующим образом персонала, связанного с ПО, на всех уровнях Министерства обороны.
- Министерство обороны не полностью определило «за» и «против», касающиеся использования коммерческих компонентов.
- Министерство обороны не уделяет достаточного внимания архитектуре.
- Министерство обороны не способствует адекватному технологическому взаимодействию с коммерческим сектором.

В отчете утверждается, что, хотя Министерство обороны и изучило множество проектов по созданию ПО (перечислены 18 проектов), большинство рекомендаций, полученных в результате этого изучения, остаются нереализованными.

Принципиальные причины, по которым проекты Министерства обороны по созданию ПО сталкиваются с проблемами, определены следующим образом:

- Плохое определение требований
- Неадекватное управление процессом создания ПО
- Отсутствие сплоченных производственных коллективов
- Неэффективное управление субподрядчиками
- Отсутствие должного внимания к процессу
- Совершенно недостаточное внимание, уделяемое архитектуре
- Плохо описанные, неадекватно контролируемые интерфейсы
- Обновление ПО для решения проблемы дефицита аппаратуры
- Основное внимание уделяется нововведениям, а не стоимости и риску
- Ограниченные возможности по адаптации оборонных стандартов или полное их отсутствие

Даются следующие основные рекомендации:

- Использовать коммерческую практику (например, итерационную разработку и процессы с упреждающим развитием архитектуры)
- Использовать коммерческие компоненты и технологии
- Инвестировать больше средств в обучение сотрудников Министерства обороны

В отчете обсуждаются способы разрешения выявленных рисков. Он не переоценивает необходимость лучшего выполнения работы по определению требований и контролю над ними, как это делалось во многих предыдущих исследованиях, проводимых Министерством обороны. Эта тема упоминается и ей уделяется должное внимание при обсуждении наряду с другими не менее важными факторами. В отчете «Хаос» основная вина за неудачу проекта возлагается на недостаточную работу с требованиями. Министерство обороны соглашалось с этим в конце 1980-х гг., но, кажется, доросло до более взвешенной самооценки и одновременного понимания симптомов и самой болезни.

Приложение B

Модель оценки стоимости COCOMO

На сегодняшний день для оценки стоимости ПО используется несколько различных моделей. Одной из самых популярных, открытых и хорошо документированных моделей оценки стоимости ПО является COCOMO (COnstructive COst MOdel — конструктивная модель стоимости), разработанная Барри Боэмом. Эволюция COCOMO отражает то, каким образом эволюционировала экономика ПО последние 20 лет.

Ключевые моменты

- История модели COCOMO позволяет увидеть эволюцию приоритетов экономики ПО.
- Исходная модель COCOMO была хорошо приспособлена для оценки стоимости традиционных проектов по созданию ПО в 1980-х гг.
- Модель Ada COCOMO усовершенствовала исходную версию, в частности, за счет параметризированной экспоненты, которая отражала усовершенствования в современном процессе и их влияние на экономию при больших масштабах.

Оценочные уравнения COCOMO имеют следующую форму:

$$\text{Работа} = C1 * EAF * (\text{Размер})^{P1}$$
$$\text{Время} = C2 * (\text{Работа})^{P2}$$

где

Работа — количество человеко-месяцев

C1 — масштабирующий коэффициент

EAF — уточняющий фактор, характеризующий предметную область, персонал, среду и инструментарий, используемый для создания рабочих продуктов процесса

Размер — размер конечного продукта (кода, созданного человеком), измеряемый в исходных инструкциях (DSI, delivered source instructions), которые необходимы для реализации требуемой функциональной возможности

P1 — показатель степени, характеризующий экономию при больших масштабах, присущую тому процессу, который используется для создания конечного продукта; в частности, способность процесса избегать непроизводительных видов деятельности (доработок, бюрократических проволочек, накладных расходов на взаимодействие)

Время — общее количество месяцев

C2 — масштабирующий коэффициент для сроков исполнения

P2 — показатель степени, который характеризует инерцию и распараллеливание, присущие управлению разработкой ПО

B.1 COCOMO

Исходная модель COCOMO [Boehm, 1981] явилась прорывом в разработке ПО в начале 1980-х гг. — частично из-за присущей ей технологической составляющей, но прежде всего потому, что она представляла собой четко определенную схему взаимодействия для достижения соглашений и установки приоритетов, касающихся управления ценой и сроками создания ПО. Будучи наивным студентом последнего курса UCLA, я впервые столкнулся с моделью COCOMO в качестве предмета нового курса, читаемого Боэмом для выпускников. Одновременно я работал в компании TRW в качестве ведущего разработчика над проектом, который требовал большого объема программирования и для которого необходимо было спланировать и обосновать оценки стоимости и сроков создания ПО. Огромным преимуществом модели COCOMO являлось то, что она позволяла получать оценку, подводить под нее заслуживающее доверия основание, судить о ее сильных и слабых сторонах и вести переговоры с заинтересованными сторонами. С тех пор я использовал модель COCOMO для обоснования технологических нововведений, усовершенствований процесса, изменений в архитектуре проекта и новых подходов к управлению. В процессе этой работы я стал экспертом в области ее сильных и слабых сторон и сфер ее применения.

Исходная модель COCOMO основывалась на базе данных по 56 проектам. Три ее варианта отражали различия между процессами в разных областях ПО. Эти варианты были описаны как обычное, частично независимое и встроенное ПО. Проекты по созданию обычного ПО характеризовались как внутренние, не очень сложные разработки, процесс осуществления которых был довольно гибким. Функциональные возможности, качества, стоимость и сроки могли свободно меняться, при этом накладные расходы оказывались минимальными. Системы, относящиеся к разряду встроенных, представляли собой типичную группу проектов для обороны: преобладающими проблемами были вопросы сложности, надежности и работы в реальном времени, а контрактная природа этого бизнеса приводила к строгому процессу. Функциональные возможности, качества, стоимость и сроки жестко контролировались, а изменения требовали согласования с множеством заинтересованных сторон. Промежуточный вариант частично независимых проектов находится посредине между двумя этими вариантами.

Формулы для оценки основных работ и сроков

Исходные соотношения для оценки стоимости в модели COCOMO имели следующий вид:

Обычный вариант

$$\text{Работа} = 3.2*\text{EAF}*(\text{Размер})^{1.05}$$
$$\text{Время (в месяцах)} = 2.5*(\text{Работа})^{0.38}$$

Промежуточный вариант

$$\text{Работа} = 3.0*\text{EAF}*(\text{Размер})^{1.12}$$
$$\text{Время (в месяцах)} = 2.5*(\text{Работа})^{0.35}$$

Встроенный вариант

$$\text{Работа} = 2.8*\text{EAF}*(\text{Размер})^{1.2}$$
$$\text{Время (в месяцах)} = 2.5*(\text{Работа})^{0.32}$$

где

Работа — количество человеко-месяцев

EAF — результат учета 15 уточняющих факторов (см. таблицу В.1)

Размер — число исходных инструкций конечного продукта (измеряемое в тысячах строк кода)

Множитель, который является уточняющим фактором работ (effort adjustment factor, EAF), представляет собой комбинацию многих параметров. Эти параметры позволяют характеризовать и нормировать проекты относительно проектов, находящихся в базе данных COCOMO. Каждый параметр оценивается как очень низкий, низкий, номинальный, высокий или очень высокий. Влияние значения каждого параметра определяется как множитель, который обычно изменяется в диапазоне от 0.5 до 1.5. Результат учета этих 15-ти параметров и используется в качестве коэффициента в уравнении стоимости.

Допущения

В формулах модели COCOMO используются некоторые допущения. Исходные инструкции конечного продукта включают в себя все (кроме комментариев) строки кода, обрабатываемого компьютером. Начало жизненного цикла проекта совпадает с началом разработки продукта, окончание — совпадает с окончанием приемочного тестирования, завершающего стадию интеграции и тестирования. (Работа и время, затрачиваемые на анализ требований, оцениваются отдельно как дополнительный процент от разработки в целом.) Виды деятельности включают в себя только работы, направленные непосредственно на выполнение проекта, в них не входят обычные вспомогательные виды деятельности, такие как административная поддержка, техническое обеспечение и капитальное оборудование. Человеко-месяц состоит из 152 часов. Проект управляется надлежащим образом, в нем используются стабильные требования.

Таблица В.1.
Параметры модели COCOMO, характеризующие проект

Идентификатор	Уточняющий фактор работ	Диапазон изменения параметра	Потенциальное влияние
RELY	Требуемая надежность	0.75 — 1.40	1.87
DATA	Размер базы данных	0.94 — 1.16	1.23
CPLX	Сложность продукта	0.70 — 1.65	2.36
TIME	Ограничение времени выполнения	1.00 — 1.66	1.66
STOR	Ограничение объема основной памяти	1.00 — 1.56	1.56
VIRT	Изменчивость виртуальной машины	0.87 — 1.30	1.49
TURN	Время реакции компьютера	0.87 — 1.15	1.32
ACAP	Способности аналитика	1.46 — 0.71	2.06
AEXP	Знание приложений	1.29 — 0.82	1.57
PCAP	Способности программиста	1.42 — 0.70	2.03
VEXP	Знание виртуальной машины	1.21 — 0.90	1.34
LEXP	Знание языка программирования	1.14 — 0.95	1.20
MODP	Использование современных методов	1.24 — 0.82	1.51
TOOL	Использование программных инструментов	1.24 — 0.83	1.49
SCED	Требуемые сроки разработки	1.23 — 1.10	1.23

Описание жизненного цикла

Жизненный цикл в модели COCOMO состоит из пяти основных стадий: планирование и определение требований, проектирование продукта, детальное проектирование, кодирование и тестирование отдельных модулей, интеграция и тестирование. Модель COCOMO дает рекомендации по распределению работ и времени по основным стадиям традиционной «водопадной» модели. Эти рекомендации в некоторой степени зависят от варианта и масштаба; в таблице В.2 приводится стандартное распределение для большого встроенного проекта. Модель COCOMO позволяет оценить работу и время, необходимые для получения решения (разработки продукта с помощью итераций и тестирования). Затраты на формулировку проблемы (планирование и определение требований) оцениваются в виде дополнительного процента от и сверх работы и времени, затрачиваемых на разработку.

Таблица В.2.
Распределение работ и времени по стадиям жизненного цикла при традиционном походе

Вид деятельности	Работа (%)	Время (%)
Планирование и определение требований	(+8)	(+36)
Проектирование продукта	18	36
Детальное проектирование	25	18
Кодирование и тестирование отдельных модулей	26	18
Интеграция и тестирование	31	28

Декомпозиция работ по созданию ПО

Декомпозиция работ при традиционном подходе обычно строится вокруг подсистем продукта на более высоких уровнях и вокруг подробного плана работ на более низких. К стандартным видам деятельности, оцениваемым по модели COCOMO и включаемым в WBS по созданию ПО, относятся: анализ требований, проектирование продукта, программирование, планирование тестирования, верификация и аттестация, канцелярские функции проекта (управление и администрирование), управление конфигурацией, обеспечение качества и создание руководств. Модель COCOMO также рекомендует самый верхний уровень распределения работ по различным видам деятельности WBS. Еще раз обращаем внимание на то, что эти значения зависят от варианта и масштаба. В таблице В.3 показаны примерные ожидаемые затраты на каждый вид деятельности WBS для большого проекта (вариант встроенного ПО). Важное предостережение: в модели COCOMO вид деятельности, называемый «программированием», включает в себя детальное проектирование, кодирование, тестирование модулей и интеграцию.

Таблица В.3.
Стандартное распределение работ по видам деятельности WBS в модели COCOMO

Вид деятельности	Бюджет (%)
Анализ требований	4
Проектирование продукта	12
Программирование	44
Планирование тестирования	6
Верификация и аттестация	14
Канцелярия проекта	7
Управление конфигурацией и обеспечение качества	7
Создание руководств	6

Типичный вид проекта в модели COCOMO

Приведем пример конкретного проекта, чтобы проиллюстрировать скрытый смысл планирования разработки. Представим себе большую, рассчитанную на 100 000 строк исходного кода (100-KDSI), критически важную систему (например, для управления электростанцией), создаваемую по контракту с правительственной организацией. На рис. B.1 показан вид проекта, оцененный с помощью модели COCOMO. Эта оценка составляет 900 человеко-месяцев на разработку плюс 72 человеко-месяца на определение требований для данного проекта. Необходимое для выполнения проекта время составит 22 месяца от начала разработки до тестирования плюс 8 месяцев на определение требований.[1]

B.2 МОДЕЛЬ ADA COCOMO

В середине 1980-х гг. компания TRW столкнулась с проблемой перевода нескольких проектов на язык Ada. В одних случаях движущей силой этого перевода служил правительственный заказ. (В этих проектах, как правило, возникало немало трудностей.) В других случаях разработчики проекта были уверены, что технология Ada окажется критичной для конкурентоспособной цены и успешного выполнения. (А эти проекты завершались успешно.) Первый прототип модели Ada COCOMO был создан мной в рамках внутреннего проекта по исследованию и разработке в 1985 г. Целью этой работы являлось создание схемы, которая позволила бы убедить руководство компании TRW и правительственных заказчиков в том, что финансовый выигрыш от применения языка Ada для определенных широкомасштабных проектов будет весьма значительным и что предложение использовать язык Ada в этих проектах является конкурентоспособной стратегией. Данный подход был также наименее рискованным, обеспечивающим создание системы в срок в рамках бюджета и требуемого качества. (Речь идет о проекте CCPDS-R — система оповещения о запусках ракет очередного поколения, — который представлен в качестве практического примера в приложении D.)

Изначально разработка модели Ada COCOMO была лишь одним из направлений выполнения триединой задачи:

1. Разработать набор компонентов на языке Ada для создания архитектуры, которая позволила бы измерить быстродействие компилятора и получить базовый комплект повторно используемых основных компонентов для управляющих и контролирующих систем типа CCPDS-R.

2. Разработать описание процесса следующего поколения, который бы использовал итерационный метод и подход с упреждающей разработкой архитектуры, основанный на демонстрациях. Эта Ada-модель процесса [Royce, Walker, 1990b] была огромным шагом вперед по направлению к применению современного процесса в проектах, имеющих отношение к обороне.

Пример: проект размером 100 000 SLOC, который требует 972 человеко-месяца работы со сроком выполнения 30 месяцев

Работа

$= 2.8*EAF*(KDSI)^{1.2}$

$= 2.8*(1.28)*(100)^{1.2}$

= 900 человеко-месяцев на разработку

+ 72 человеко-месяца на планирование, определение требований

= 972 человеко-месяца суммарно

Время

$= 2.5*(\text{Работа})^{0.32}$

$= 2.5*(900)^{0.32}$

= 22 месяца на разработку

+ 8 месяцев на планирование, определение требований

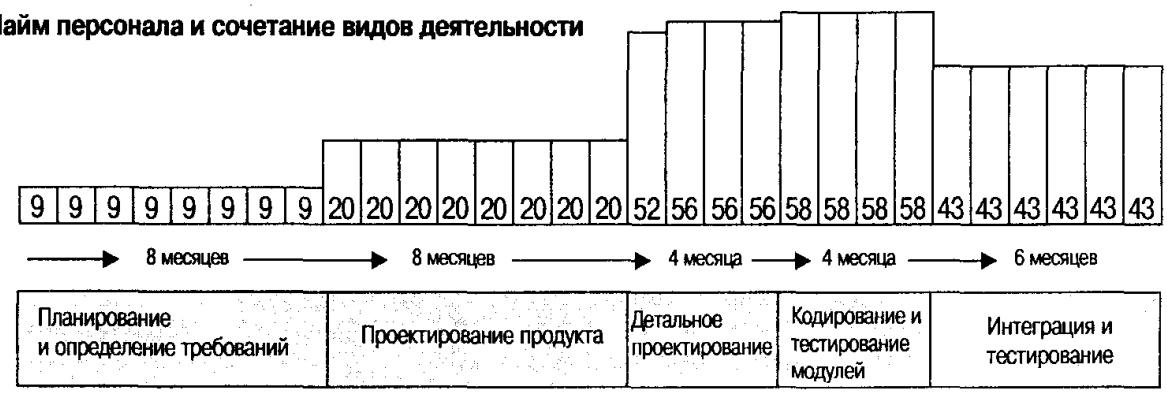
= 30 месяцев

Общий EAF (в данном случае — 1.28) является производным от отдельных составляющих, влияющих на определение затрат.

Фактор, влияющий на стоимость	Значение	Влияние
Знание языка программирования	Номинальное	1.0
Ограничение времени выполнения	Номинальное	1.0
Размер базы данных	Номинальное	1.0
Межремонтный срок службы компьютера	Номинальное	1.0
Знание виртуальной машины	Номинальное	1.0
Изменчивость виртуальной машины	Номинальное	1.0
Использование программных инструментов	Высокое	0.88
Использование современных методов	Номинальное	1.0
Ограничение объема памяти	Номинальное	1.0
Знание приложений	Низкое	1.10
Ограничение по срокам разработки	Номинальное	1.0
Требуемая надежность	Высокое	1.15
Сложность продукта	Высокое	1.15
Способности персонала/команды	Номинальное	1.0

Уточняющий фактор работ = 1.28

Найм персонала и сочетание видов деятельности



Распределение времени и работ по жизненному циклу в модели COCOMO изменяется в зависимости от масштаба, области применения и условий. Приведенные здесь время, отводимое на различные стадии, и сочетание разных видов деятельности являются типичными.

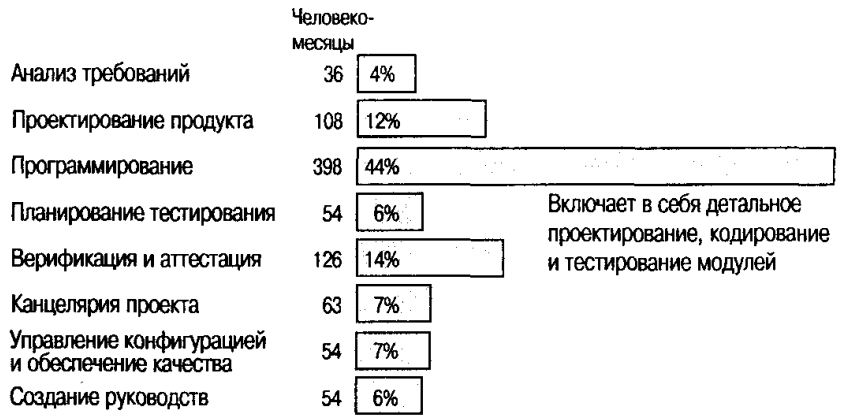


Рис. В.1. *Общий вид оценки традиционного проекта*

3. Разработать Ada-версию модели COCOMO для того, чтобы описать выигрыш в затратах денег и времени для новых технологии и процесса.

Результаты этой работы оказались критичными для подхода компании TRW к проекту CCPDS-R, а создание модели Ada COCOMO стало ключевым при продаже подхода в целом как менеджерам, так и правительственному заказчику. Первоначальная версия была впоследствии формализована в рамках компании TRW под руководством Боэма [Boehm and Royce, Walker, 1988]. Был учтен опыт нескольких других проектов, определены значения параметров и расширена область применения за счет введения понятия параметризованной экспоненты.

Первоочередным усовершенствованием в модели Ada COCOMO была попытка избавиться от трех вариантов модели COCOMO (обычная, промежуточная, встроенная) и сделать экспоненту параметризованной, что позволило бы отразить экономию при больших масштабах, присущую современному итерационному процессу разработки. Некоторые менее значительные доработки были призваны адаптировать другие параметры под преимущества, которыми обладала Ada-среда.

Соотношения для оценки стоимости в модели Ada COCOMO имели следующий вид:

$$\text{Работа} = 2.8 * \text{EAF} * (\text{Размер})^{P}$$
$$\text{Время} = 2.5 * (\text{Работа})^{0.32}$$

где

Работа — количество человеко-месяцев

EAF — результат учета 19 уточняющих факторов работы (см. таблицу В.4)

Размер — число исходных инструкций конечного продукта (измеряемое в тысячах строк кода)

P — показатель степени

Время — общее количество месяцев

Множитель EAF по-прежнему представляет собой комбинацию нескольких параметров. Однако в модель Ada COCOMO были внесены некоторые изменения, отражающие общее усовершенствование модели COCOMO, эффекты, специфичные для языка Ada, и эффекты от применения итерационного процесса. Это уточнение привело к добавлению двух новых стоимостных факторов (RUSE и SECU), раздвоению одного из факторов (VIRT был поделен на два компонента: относящийся к хосту и к целевой машине) и к появлению нескольких новых диапазонов значений или к изменениям в эффектах, лежащих в основе стоимостных факторов.

Одним из обоснований модели Ada COCOMO было применение Ada-модели процесса. Для использования основных методов этой модели процесса не было необходимости привлекать язык Ada. Однако на момент разработки модели на рынке оборонного ПО было столько спекуляций как переоценивающих, так и недооценивающих этот язык, что описание процесса было привязано к применению Ada. Такой подход имел

свои «за» и «против». В ретроспективе Ada-модель процесса может рассматриваться как промежуточное состояние между схемами традиционного процесса и современного процесса, описанного в настоящей книге. Для того чтобы понять параметризацию процесса, присущую экспоненте модели Ada COCOMO, приведем обобщение различных стратегий Ada-модели процесса.

Таблица B.4.
Усовершенствование уточняющих факторов работы для модели Ada COCOMO

Идентификатор	Уточняющий фактор работы	Изменения для модели Ada COCOMO
RELY	Требуемая надежность	Изменения в базовых эффектах (положительное влияние)
DATA	Размер базы данных	Без изменений
CPLX	Сложность продукта	Изменения в базовых эффектах (положительное влияние)
RUSE	Требуемый уровень повторного использования	Новый эффект для учета сложности компонентов повторного использования
SECU	Ограничения по безопасности	Новый эффект для секретных проектов
TIME	Ограничение времени выполнения	Без изменений
STOR	Ограничение объема основной памяти	Без изменений
VIRT	Изменчивость виртуальной машины	Исключен (разбит на два новых фактора)
VMVH	Изменчивость виртуального хоста	Новый эффект, учитывающий аспекты VIRT, связанные с хостом
VMVT	Изменчивость целевой виртуальной машины	Новый эффект, учитывающий аспекты VIRT, связанные с целевой машиной
TURN	Время реакции компьютера	Новый уровень интерактивного взаимодействия (положительное влияние)
ACAP	Способности аналитика	Изменения в базовых эффектах (большее влияние)
AEXP	Знание приложений	Без изменений
PCAP	Способности программиста	Изменения в базовых эффектах (меньшее влияние)
VEXP	Знание виртуальной машины	Без изменений
LEXP	Знание языка программирования	Изменения в базовых эффектах (большее влияние)
MODP	Использование современных методов	Изменения в базовых эффектах (большее влияние)
TOOL	Использование программных инструментов	Новые уровни автоматизированной поддержки
SCED	Требуемые сроки разработки	Изменения в базовых эффектах (меньшее влияние)

Одной из критичных стратегий Ada-модели процесса было выделение контрольной точки обзора предварительной разработки (PDR, preliminary design review), которая требовалась в соответствии с применяемым военным стандартом как рассмотрение архитектуры, подкрепленное демонстрацией возможностей. Эта объемлющая цель приводила к нескольким частным стратегиям, которые задействовали способы, инструментарий и технологии Ada-среды:

- Небольшое ядро команды разработчиков, обладающих опытом и знаниями в области архитектуры ПО и в прикладной области
- Внимание на ранних стадиях исполняемому «скелету» архитектуры для демонстрации критичных компонентов и системных потоков управления с целью определения рисков
- Пошаговый и независимый сквозной контроль компонентов и версий вместо полного критического обзора проекта сразу для всей системы
- Постоянная интеграция посредством Ada-компиляции и упреждающая разработка архитектуры
- Основное внимание при тестировании программы и верификации требований уделяется последовательности тестов (которые теперь называются вариантами использования) и независимому тестированию отдельных компонентов
- Самодокументируемый код на языке Ada и общие описания вместо толстых подробных проектных документов, в которых разработка описывается по мере ее развития
- Метрики, автоматически извлекаемые из кода

Показатель степени в модели Ada COCOMO имеет диапазон от 1.04 до 1.24 и описывается как комбинация следующих четырех параметров:

1. Знакомство с Ada-моделью процесса. Этот уровень зрелости процесса изменяется от значения «знакомство отсутствует» до «успешное применение во многих проектах».
2. Тщательность проектирования, предшествующего PDR. Этот параметр характеризует уровень детализации проекта, присущий базовому проекту, демонстрируемому при PDR. Он изменяется в диапазоне от «недостаточной тщательности» (20%) до «абсолютной тщательности» (100%).
3. Риски, от которых удалось избавиться при PDR. Этот параметр оценивает уровень неопределенности, присущий проекту при PDR, после которого начинается полномасштабная разработка. Он изменяется в диапазоне от «недостаточного разрешения рисков» (20%) до «полного разрешения» (100%).
4. Изменчивость требований в процессе разработки. Этот параметр, изменяющийся в диапазоне от «очень большие изменения» до «изменения отсутствуют», характеризует количество возмущений, с которыми сталкивается проект.

Реальный показатель степени в модели Ada COCOMO определяется суммой значений всех параметров, нормированных от 0.00 до 0.05. Показатель степени для встроенного ПО (1.20) в исходной модели COCOMO будет соответствовать Ada-модели процесса с значением каждого параметра процесса, равным 0.04 [1.04 + (4 x 0.04)]. В терминах описанных выше параметров это будет соответствовать (1) слабому знакомству с Ada-моделью процесса, (2) некоторой тщательности разработки на стадии PDR (40%), (3) определенному количеству рисков, от которых удалось избавиться к PDR (40%), и (4) частым, но умеренным изменениям требований.

Эти параметры нужны прежде всего для того, чтобы характеризовать способность процесса к снижению платы за большой масштаб, присущей традиционному процессу. Поддерживая размер команды на низком уровне и создавая более осязаемое описание архитектуры для PDR, процесс пытается оптимизировать межличностные взаимодействия, избежать на поздних этапах переделки того, что создано на более ранних, и способствовать раннему определению окончательных требований.

B.3 COCOMO II

Проект COCOMO II [Boehm и другие, 1995; Horowitz, 1997] – это работа, которая выполнялась в Центре по разработке ПО USC (USC Centre for Software Engineering) с финансовой и технической поддержкой огромного количества промышленных предприятий. (В их число входили AT&T Bell Labs, Bellcore, DISA, EDS, E-Systems, Hewelett-Packard, Hughes, IDA, IDE, JPL, Litton Data Systems, Lockheed Martin, Loral, MDAC, Motorola, Northrop-Grumman, Rational, Rockwell, SAIC, SEI, SPC, TASC, Teledyne, Texas Instruments, TRW, USAF Rome Labs, US Army Research Lab и Xerox.) Проект имел триединую задачу:

1. Разработать модель для оценки стоимости и сроков создания ПО для того жизненного цикла, который будет применяться в 1990-х и 2000-х гг.

2. Разработать базу данных по стоимости ПО и инструментальную поддержку для усовершенствования модели стоимости.

3. Создать количественную аналитическую схему для оценки технологий создания ПО и их экономического эффекта.

USC предполагает, что после 2000 г. на рынке ПО будут присутствовать:

1. Программисты приложений для конечных пользователей (55 млн.), которые создают электронные таблицы или запросы к базам данных.

2. Разработчики компонентов (600 тыс.), которые создают приложения для конечных пользователей и вспомогательные средства.

3. Интеграторы компонентов (700 тыс.), которые быстро строят приложения, используя готовые библиотеки GUI, системы управления объектами/базами данных, промежуточное ПО и специализированные для данной области компоненты.
4. Системные интеграторы (700 тыс.), которые работают с системами большего масштаба, с системами, не имеющими аналогов, с приложениями, имеющими ограниченное число аналогов, с встроенными системами, требующими предварительной разработки, и с другими значительными разработками ПО на заказ.
5. Разработчики инфраструктуры (750 тыс.), которые разрабатывают не зависящие от области применения компоненты, такие как операционные системы, системы управления базами данных, сети и структуры пользовательских интерфейсов.

Конечные пользователи не являются объектом изучения модели COCOMO II, поскольку они обычно производят скоротечные, небольшие работы, для которых простые оценки, основанные на видах выполняемых работ, оказываются вполне достаточными. Основным целевым рынком для моделей оценки стоимости ПО являются коллективы, состоящие из нескольких человек и работающие в течение месяцев или даже лет.

Стратегия модели COCOMO II направлена на сохранение открытости исходной модели COCOMO, ее адаптацию к описанному выше рынку, приведение входной и выходной информации в соответствие доступному уровню информации и создание возможностей для адаптации модели к различным стратегиям выполнения проектов. В частности, это поколение моделей COCOMO использует диапазон оценок вместо точечных оценок. Они изменяются на протяжении всего жизненного цикла от грубой входной информации и оценок в широких диапазонах на ранних стадиях до более точной входной информации и более точных оценок на поздних стадиях. На рис. В.2 показано изменение точности оценок на протяжении жизненного цикла.

Для поддержки такой стратегии в рамках модели COCOMO определяются три различные модели оценки стоимости. На рис. В.3 показано соответствие этих трех моделей стадиям итерационного жизненного цикла. Модели соответствуют уровню правильности и неопределенности, присущему текущей стадии жизненного цикла. Постархитектурная модель почти полностью соответствует модели COCOMO, в которой принято допущение о том, что проект имеет стабилизировавшиеся требования, планы и начальное представление о предполагаемой архитектуре. Затем проекты до конца следуют «водопадному» процессу с практически неизменными требованиями. Постархитектурная модель предназначена для получения точных оценок после того, как у проекта появляются базовые требования, базовая архитектуры и план на стадию конструирования. Модель ранних этапов проектирования дает более грубые оценки на той стадии жизненного цикла, на которой происходит уточнение, а модель компоновки приложений позволяет получать очень грубые по порядку величины оценки в стадии начала проекта.

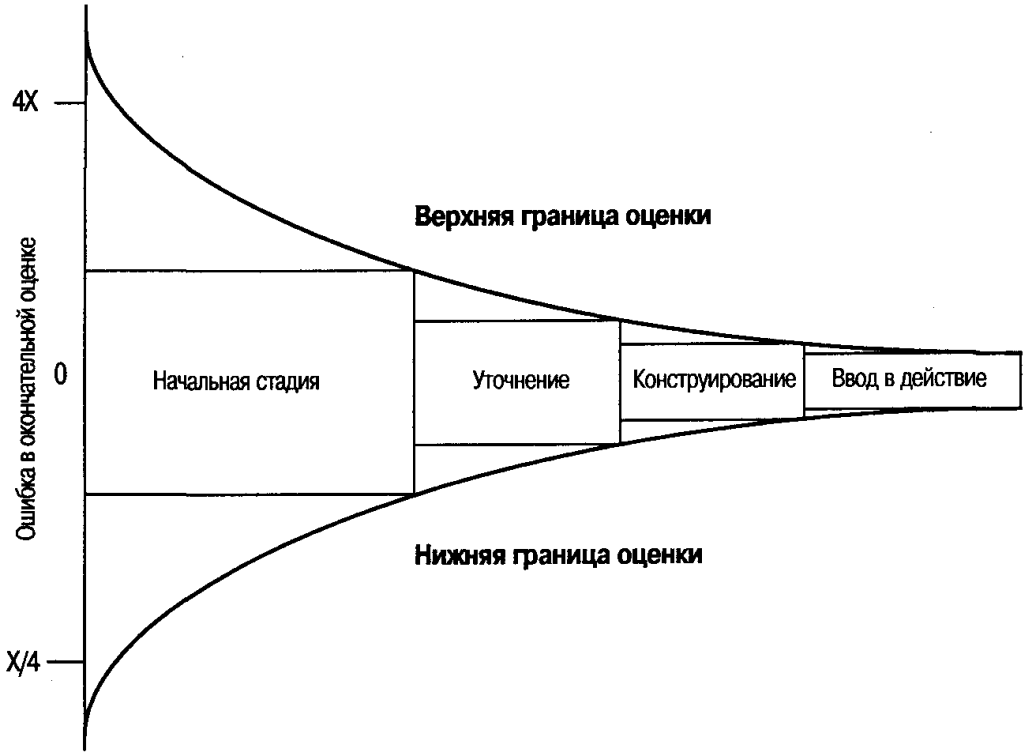
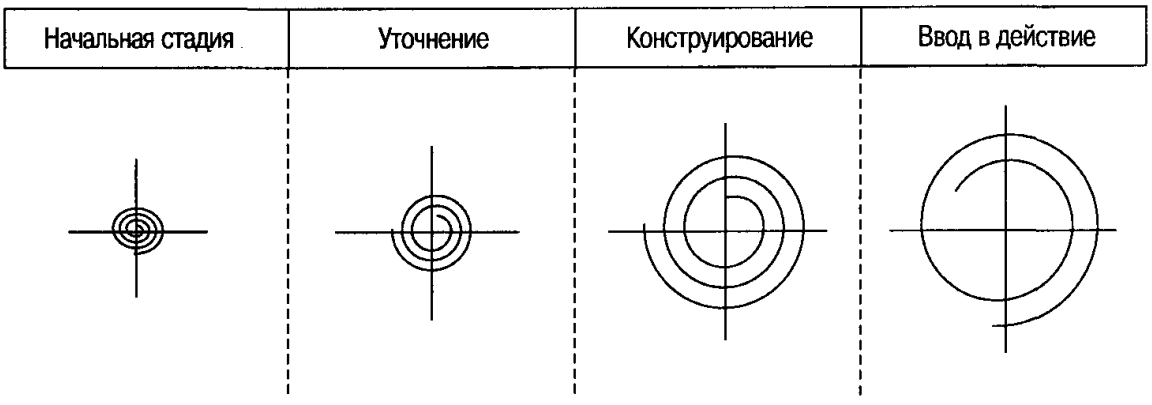


Рис. В.2. *Изменение оценки ПО на протяжении жизненного цикла*



Модели оценки стоимости COCOMO II		
Модель этапа создания прототипов	Модель ранних этапов разработки	Постархитектурная модель
Грубые входные данные	Ясно понимаемые особенности проекта	Детальное описание проекта
Оценки низкой точности	Оценки умеренной точности	Высокоточные оценки
Приблизительные требования	Ясно понимаемые требования	Стабилизировавшиеся основные требования
Концепция архитектуры	Ясно понимаемая архитектура	Стабильная базовая архитектура

Рис. В.3 *Оценки с помощью модели COCOMO II на протяжении всего жизненного цикла*

Модель компоновки приложений соответствует исследовательской работе, обычно выполняемой в процессе создания прототипов и анализа осуществимости. Оценочное уравнение представляет собой простое линейное соотношение между объектными точками и сложностью данной области.

Модель ранних этапов проектирования соответствует уровню детализации, достижимому на стадии разработки проекта, в течение которого происходит синтез архитектуры, требований и планов. Общее уравнение оценки стоимости имеет вид:

$$\text{Работа} = 2.45 * E_{Arch} * (\text{Размер})^{P}$$

где

Работа — число человеко-месяцев

E_{Arch} — результат применения семи уточняющих факторов ранних этапов проектирования (см. таблицу В.5)

Размер — число функциональных точек (предпочтительно) или KSLOC

P — показатель степени

Таблица В.5.
Уточняющие факторы на ранних этапах

Идентификатор	Составные уточняющие факторы работы
Сложность продукта	RELY-DATA-CPLX-DOCU
Необходимость повторного использования	RUSE
Сложность платформы	TIME-STOR-PVOL
Опытность персонала	AEXP-PEXP-LTEX
Способности персонала	ACAP-PCAP-PCON
Возможности	TOOL-SITE
Сроки	SCED

Параметры ранней стадии проектирования формируются из параметров постархитектурной фазы. Они позволяют применять более простой оценочный метод на ранних стадиях жизненного цикла, когда еще довольно много неизвестных.

Уравнение для оценки стоимости в постархитектурной фазе имеет вид:

$$\text{Работа} = 2.45 * E_{App} * (\text{Размер})^{P}$$

где

Работа — число человеко-месяцев

E_{App} — результат применения семнадцати уточняющих факторов постархитектурных этапов разработки (см. таблицу В.6)

Размер - количество KSLOC (предпочтительно) или число функциональных точек

P - показатель степени

Таблица В.6.
Усовершенствованная по сравнению с Ada COCOMO и COCOMO постархитектурная модель COCOMO II

Идентификатор	Уточняющий фактор работы	Изменения в COCOMO II
RELY	Требуемая надежность	Без изменений (относительно COCOMO)
DATA	Размер базы данных	Без изменений (относительно COCOMO)
CPLX	Сложность продукта	Без изменений (относительно COCOMO)
RUSE	Требуемый уровень повторного использования	Без изменений (относительно Ada COCOMO)
DOCU	Документация	Добавлен; определяет, насколько документация соответствует требованиям жизненного цикла
TIME	Ограничение времени выполнения	Без изменений (относительно COCOMO)
STOR	Ограничение объема основной памяти	Без изменений (относительно COCOMO)
PVOL	Изменчивость платформы	Фактор изменчивости платформы является комбинацией параметров VMVH и VMVT модели Ada COCOMO
ACAP	Способности аналитика	Без изменений (относительно COCOMO)
AEXP	Знание приложений	Без изменений (относительно COCOMO)
PCAP	Способности программиста	Без изменений (относительно COCOMO)
PEXP	Знание платформы	Параметр, определяющий знание платформы, является расширением параметра знания виртуальной машины
PCON	Преемственность персонала	Новый параметр
LTEX	Знание языка/инструментария	Изменен с целью охвата знаний и инструментария, и языка
SITE	Распределенная разработка Взаимодействие между командами разработчиков	Новые параметры, определяющие степень взаимной удаленности команд разработчиков и степень автоматизации их деятельности
TOOL	Использование программных инструментов	Без изменений (относительно COCOMO)
SCED	Требуемые сроки разработки	Без изменений (относительно COCOMO)

Коэффициенты E отражают совместное влияние многих параметров. В постархитектурной модели применяются параметры, аналогичные используемым в традиционной модели COCOMO. Эти параметры позволяют характеризовать и нормировать среду разработки по параметрам, содержащимся в базе данных проектов модели COCOMO II (в настоящее время 83 проекта). Каждый параметр в зависимости от установленного значения (очень низкое, низкое, номинальное, высокое, очень высокое) вносит свой вклад в виде множителя с диапазоном значений обычно от 0.5 до 1.5. Результат учета этих 17 эффектов и используется при вычислении работы в уравнении стоимости.

Из самого названия постархитектурной модели становится ясно, что она описывает продукт, получаемый на ранней стадии проектирования, т.е. архитектуру. Для количественного определения размера на ранней стадии проектирования рекомендуется использовать функциональные точки, поскольку именно функциональные точки лучше приспособлены к ранним стадиям, когда структура (а, следовательно, и оценки в SLOC) конечного продукта окончательно не ясна. Использование SLOC рекомендовано для количественного определения размера в постархитектурной модели. Такой подход – неплохой технический компромисс между апологетами SLOC и апологетами функциональных точек.

Модель COCOMO II использует одну и ту же показательную функцию для моделей ранних этапов проектирования и для постархитектурных моделей. Показатель степени может изменяться в диапазоне от 1.01 до 1.26 и определяется как сочетание влияния следующих параметров:

1. Наличие прецедентов у приложения: уровень опыта организации-разработчика в данной области
2. Гибкость процесса: степень строгости контракта, порядок его выполнения, присущая контракту свобода внесения изменений, виды деятельности в течение жизненного цикла и взаимодействие между заинтересованными сторонами
3. Разрешение рисков, присущих архитектуре: степень технической осуществимости, продемонстрированной до перехода к полномасштабному производству
4. Сплоченность команды: степень сотрудничества и того, насколько все заинтересованные стороны (покупатели, разработчики, пользователи, ответственные за сопровождение и др.) разделяют общую концепцию
5. Зрелость процесса: уровень зрелости организации-разработчика, определяемая в соответствии с моделью CMM

Параметризация экспоненты в модели COCOMO II является эволюционным развитием подхода, использованного в модели Ada COCOMO, на более солидной основе. В таблице B.7 обобщены возможные значения параметров. Реальный показатель экспоненциальной функции модели COCOMO II определяется как сумма эффектов всех параметров. Объединенное влияние этих параметров процесса может оказаться весьма существенным. Вместе с тем команда разработчиков модели COCOMO II допускает реальную экономию при больших масштабах (это означает, что значение P никогда не становится меньше единицы). Они убеждены в том, что экономия при больших масштабах может быть достигнута за счет сокращения размера в результате использования коммерческих компонентов, компонентов повторного использования, CASE-средств и объектно-ориентированных технологий.

Таблица B.7.
Параметры показателя экспоненциальной функции модели COCOMO II

Параметр	Очень низкое (0.00)	Низкое (0.01)	Номинальное (0.02)	Высокое (0.03)	Очень высокое (0.04)	Чрезвычайно высокое (0.05)
Наличие прецедентов	Полное отсутствие прецедентов	Почти полное отсутствие прецедентов	Наличие некоторого количества прецедентов	Общее знакомство	Широкое знакомство	Исчерпывающее знакомство
Гибкость р азработки	Строгая	Случайные послабления	Некоторые послабления	Общее соответствие	Некоторое соответствие	Общие цели
Разрешение рисков в архитектуре	Малое 20%	Некоторое 40%	Частое 60%	В целом 75%	Почти полное 90%	Полное 100%
Сплоченность команды	Сильно затрудненное взаимодействие	Несколько затрудненное взаимодействие	Некоторая согласованность	Повышенная согласованность	Высокая согласованность	Взаимодействие как единого целого
Зрелость процесса	Уровень 1	Уровень 2	Уровень 2+	Уровень 3	Уровень 4	Уровень 5

Еще одно интересное усовершенствование в модели COCOMO II – уравнение для оценки сроков, которое теперь является функцией как оценки работы, так и параметров процесса. Результирующее влияние лучшего процесса приводит к уменьшению как работы, так и сроков.

В целом модель COCOMO II является усовершенствованием традиционных моделей стоимости, многие из которых давно устарели. Она хорошо соответствует итерационной разработке, современной технологии и процессу управления, описанному в данной книге. Вместе с тем она является незрелой, и в ее базу данных по-прежнему поступают разнообразные проекты из бесчисленных организаций. Трудно поверить в то, что она сможет оказаться более надежной, чем модель COCOMO.

Приложение C

Метрики изменений

Ключевые моменты

- Одной из наиболее важных характеристик хорошего ПО является легкость внесения в него изменений.
- Измеряя и оценивая объем работы по дефектам и доработкам при последовательном создании базовых версий ПО, можно добиться понимания того, приближается ли процесс к приемлемым уровням качества и прогресса, или удаляется от них.
- Метрики, извлекаемые непосредственно из технических рабочих продуктов, являются основой для использования инструментальных средств, способствующего согласованному, аккуратному и точному контролю над проектом.

Измерение прогресса в создании ПО и его качества является чрезвычайно трудной задачей по причине наличия огромного количества параметров продукта, проекта и персонала, влияющих на разработку ПО. Практически невозможно сформулировать абсолютные определения мер, которые бы удовлетворяли всем проектам. Однако некоторые аспекты измерения ПО являются общими и могут быть применены к большинству проектов.

В основе данного подхода к определению метрик лежит гипотеза:

> *Наиболее важной характеристикой ПО является возможность его изменения: чем проще вносить изменения в ПО, тем проще достигнуть всех остальных необходимых характеристик.*

Основные метрики, таким образом, связаны с тенденциями изменения ПО (дефектами и доработками) во всех рабочих продуктах на протяжении всего жизненного цикла. Для того чтобы иметь возможность управлять наиболее важными работами по созданию ПО, менеджеру проекта требуется несколько контекстно-независимых (для сравнения с общими потребностями) и несколько контекстно-зависимых метрик.

Большинство из этих материалов было разработано мной в 1987 г. с целью обоснования программы метрик для проекта CCPDS-R (см. приложение D). Данный материал был опубликован [Royce, Walker, 1990] после

того, как за три года практического применения была продемонстрирована его полезность и были внесены некоторые улучшения. За последние двадцать лет предпринимались многочисленные попытки измерить качество ПО. По многим причинам ни одна из них не прижилась на практике, хотя в них присутствует несколько моментов, практически полностью совпадающих с моими рекомендациями. К постоянно встречающимся трудностям относятся субъективность подхода и затраты на сбор и интерпретацию значений этих метрик.

C.1 ОБЩИЙ ОБЗОР

Мой подход к метрикам аналогичен подходу ДеМарко, который предложил определять качество ПО как степень отсутствия в нем брака [DeMarco, 1982]. Для того чтобы можно было использовать его подход вне зависимости от технологии и проекта, его положения преднамеренно неопределенны, мои же — совершенно точны. Согласованное применение важно для правильной интерпретации так же, как в случае со способами оценки стоимости. Оценка стоимости ПО имеет на входе субъективные данные, а на выходе — объективные. Мой подход определяет объективные входные данные, которым потребуется субъективная интерпретация в контексте конкретного проекта.

Одним из эффективных способов оценки качества ПО на протяжении жизненного цикла является измерение количества доработок в базовых версиях ПО. В качестве единицы измерения могут использоваться строки исходного кода, функциональные точки, объектные точки, файлы, компоненты или какие-либо другие меры объема ПО. В качестве основной единицы измерения мы будем использовать SLOC, поскольку они наиболее распространены в промышленности, являются мерой, простейшей для понимания, и лучше всего соответствуют данным практического примера в приложении D.

В некоторых случаях оценка качества ПО, полученная на основе множества объективных метрик изменений, может потребовать контекстно-зависимой интерпретации. Для оценки качества посредством любой метрики необходимо обсуждение. Одни и те же метрики следует использовать при оценке качества в процессе разработки (для определения тенденций) и по окончании разработки (для определения значений). Например, объем доработок после поставки продукта заказчику является объективной мерой качества или отсутствия такового. Количество доработок, последовавшее после создания первой базовой версии в процессе разработки, является неоднозначным вне контекста последующих действий. Отсутствие доработок может быть проинтерпретировано как очень хорошая основа (что маловероятно), как неадекватность тестирующей программы или как непритязательность первоначальной разработки.

Качество ПО

Чрезвычайно сложно объективно описать это понятие. Существуют всего лишь два механизма, позволяющих определить ожидаемое заказчиком качество: выяснение требований к возможностям и работе ПО и

утверждение плана расходов, в котором зафиксированы затраты и сроки. Такие материалы, обычно относящиеся к контракту, как правило, имеют самое низкое качество из всех материалов, создаваемых в рамках проекта, поскольку их необходимо согласовывать в самом начале жизненного цикла, когда еще присутствует слишком много неизвестных. Современный итерационный процесс и объективные параметры ПО обеспечивают лучшее понимание той степени, в которой функциональные возможности, производительность, стоимость и сроки соответствуют ожиданиям заказчика.

Запросы на внесение изменений

SCO (см. главу 12) определяют направление работ по внесению изменений в сконфигурированный программный компонент. (SCO нередко называют отчетами о проблемах ПО, однако «проблема» подразумевает нечто негативное, а далеко не все изменения предопределяются наличием проблем.) Внесение изменений может понадобиться для доработки низкокачественного компонента (типы 0 и 1, ошибка), для доработки компонента с целью повышения его качества (тип 2, улучшение) или для приведения в соответствие с изменениями в требованиях, внесенными по указанию заказчика (тип 3, изменение в области применения). Различие между ошибкой и улучшением кроется в причине, по которой вносится изменение. Если причиной внесения изменений является повышение эффективности, тестируемости, полезности и др. (предполагая, что компонент обладает достаточным качеством), доработке будет присвоен тип 2. Доработки типа 0, 1 и 2 приводят к повышению качества конечного продукта. Однако типы 0 и 1 указывают также на ненадлежащее качество имеющейся версии. На практике разграничение между типами 0, 1 и типом 2 может быть весьма субъективным. Как следует из дальнейшего обсуждения, большинство метрик не чувствительно к подобному разбиению на категории, но если такая дифференциация применяется последовательно, она может дать некоторую полезную информацию. Метрики изменений типов SCO 0, 1 и 2 собираются и анализируются совместно.

SCO типа 3 обычно отражают изменения в требованиях, которые приводят к переопределению ожиданий заказчика. Влияние таких изменений намного шире, следовательно, для них требуются различные уровни разработки ПО и систем, а также совершенно разные уровни регрессионного тестирования. По причине большого диапазона изменчивости SCO типа 3 для этих метрик анализируются отдельно. Данные, полученные из SCO типов 0, 1 и 2, обеспечивают твердое основание для оценки сопровождаемости и работы, необходимой для выполнения SCO типа 3.

Строки исходного кода

Являются ли SLOC хорошим способом измерения объема ПО, всегда было спорным вопросом. (ДеМарко называет это термином «bang» (громкий шум).) Джонс указывает на некоторые предосторожности, необходимые при работе с SLOC [Jones, 1994]. Он говорит, что «использование

строк кода для нормализации данных при наличии нескольких различных языков программирования следует рассматривать как профессиональное преступление». Пунктом, с которым согласны все, является следующее утверждение: что бы ни применялось, оно должно быть определено объективно и непротиворечиво для того, чтобы иметь ценность при выполнении сравнений. Каким именно способом дать абсолютное определение единицы измерения SLOC не так важно, как определить ее согласованным образом во всех проектах и во всех областях конкретного проекта. Необходимость в использовании единого инструмента для подсчета приводит к стандартизации на базе данного определения.

Совет по управлению конфигурацией

CCB (Configuration Control Board) — это управляющий орган, ответственный за утверждение изменений, вносимых в базовую версию продукта. В него входят как минимум менеджер по разработке ПО, менеджер по оценке ПО и — для работ, выполняемых по контракту,— представитель заказчика. CCB принимает решения по всем предлагаемым изменениям, вносимым в продукт, и утверждает все SCO. CCB ответственен за сбор параметров качества ПО, объективный и субъективный анализ тенденций и за предложения по изменению процесса разработки, инструментария, продуктов или персонала с целью улучшения качества в будущем.

Сконфигурированная базовая версия

Сконфигурированная базовая версия — это набор продуктов, внесение изменений в которые подвергается контролю со стороны CCB. Они могут представлять собой промежуточный продукт, для которого завершены проектирование, разработка и неформальное тестирование, либо конечный продукт, для которого проведено формальное тестирование.

C.2 ПОЛУЧЕНИЕ МЕТРИК

В этом разделе определяются и подробно описываются статистика, которую необходимо собирать, метрики, извлекаемые из этой статистики, и некоторые общие указания по их интерпретации. В приложении D приводятся подробные примеры их прикладного использования в качестве иллюстрации того, как эти метрики могут быть применены для управления и контроля над проектом. Получение метрик не является очевидной прямой последовательностью действий; напротив, это результат проб и ошибок, бесчисленных эмпирических анализов, интуиции и эвристик.

Исходная статистика, которую необходимо собирать, включает в себя количество и типы изменений, вносимых в ПО, число SLOC негодного кода и число SLOC исправленного кода. Проблема заключается в том, чтобы построить правильный фильтр для необработанной статистики, позволяющий выявить полезные тенденции и найти объективную количественную меру прогресса (промежуточные атрибуты в процессе разработки) и качества (атрибуты конечного продукта). Конечной целью является количественное определение коэффициента дефектности,

адаптируемости, завершенности и сопровождаемости. Коэффициент дефектности и адаптируемость можно интуитивно определить как функцию переделок; завершенность и сопровождаемость — понятия более тонкие.

- Коэффициент дефектности. Эта метрика определяет среднюю степень брака или дефектов. Она идентифицирует необходимость количественного выражения дефектов (количество негодных SLOC) и число случаев доработки (число SCO). Фактически коэффициент дефектности определяется как мера локализации брака; чем меньше его значение, тем лучше.

- Адаптируемость. Этот параметр определяет средний уровень сложности исправления дефектов, измеряемый как объем доработок. Он идентифицирует необходимость количественного выражения доработок (работы, требующейся для решения проблемы) и число случаев доработки (число SCO). Адаптируемость является количественным выражением простоты изменений; чем меньше ее значение, тем лучше.

- Завершенность. Интуитивно завершенность соответствует уровню надежности продукта. Объективно этот параметр определяет уровень дефектов. Целью является полное отсутствие дефектов, т.е. абсолютная завершенность. Доверие возрастает прежде всего благодаря широкому использованию продукта. Поскольку ПО является интеллектуальной собственностью, не имеющей материальной составляющей, оно не подвержено физическому износу. ПО со временем совершенствуется, это означает, что его пользователи (ответственная за тестирование команда, пользователи бета-версии, пользователи окончательной версии) на практике будут все реже сталкиваться с дефектами по мере появления каждой последующей версии продукта. Ожидание того, что с появлением новых версий завершенность будет возрастать, оказывается верным даже в том случае, когда изменения касаются производительности и функциональных возможностей. Аналогично, на протяжении всего жизненного цикла итерационной разработки должны быть хорошо заметны тенденции совершенствования версий. В случае простого показателя количества дефектов потребуется определение количества дефектов (SCO типов 0 и 1) и общего времени использования. На основе этих параметров можно найти среднее время наработки на отказ (MTBF) для данной версии. Чем выше значение показателя завершенности, которое соответствует среднему времени между двумя отказами в восприятии пользователя, тем лучше.

- Сопровождаемость. Теоретически сопровождаемость продукта имеет отношение к продуктивности, с которой может работать команда, ответственная за сопровождение. Однако продуктивность разных проектов настолько сложно сравнивать, что это определение кажется интуитивно неудовлетворительным. Отношение продуктивности доработок к продуктивности разработки позволяет

получить значение, не зависящее от продуктивности, но вместе с тем отражающее сложность разработки. Это отношение нормализует различия в продуктивности для разных проектов и позволяет получить метрику, подлежащую сравнению. Таким образом, сопровождаемость определяется как отношение продуктивности доработок к продуктивности разработки. Интуитивно понятно, что это значение идентифицирует продукт, эффективность изменения которого в три раза выше, чем эффективность его создания (значение сопровождаемости равно 0.33), как имеющий лучшую (более низкую) сопровождаемость, чем продукт, эффективность изменения которого всего в два раза выше (значение сопровождаемости равно 0.5) эффективности его создания, независимо от абсолютных значений реальной продуктивности сопровождения. Статистика, необходимая для вычисления этих значений,— это суммарные действия, направленные на разработку, суммарное количество SLOC, суммарные действия по доработке и суммарное количество доработанных SLOC.

В то время как эти значения являются объективной мерой конечного продукта, их промежуточные значения в виде функции времени позволяют получать в процессе разработки представление об ожидаемых для конечного продукта значениях. Опыт по сопровождению, полученный проектом на первых этапах своего осуществления, оказывается полезным для предсказания объема доработок на оставшихся этапах.

Это краткое описание относительно просто. Нет необходимости работать с полным множеством метрик, хотя разные точки зрения нужны менеджерам для аккуратного управления процессом. Подмножества или различные множества метрик также оказываются полезными. Основная часть анализа, расчетов и сбора информации, требуемых для определения этих метрик, должна быть автоматизирована, в таком случае практикам останется только интерпретировать результаты и понять, на чем они основываются.

C.2.1 Сбор статистики

Для практической работы с предложенными метриками требуется собирать некоторую специальную статистику на протяжении всего жизненного цикла проекта. Виды этой статистики приведены в таблице C.1.

- Общее число SLOC ($SLOC_T$). Этот вид статистики позволяет отслеживать оценочный объем разрабатываемого продукта. Данное значение может существенно изменяться на протяжении жизненного цикла разработки по мере того, как неизвестные, присутствовавшие в требованиях на ранних этапах, определяются, и решения, принимаемые по разработке, становятся более зрелыми. Это общее число должно включать в себя также повторно используемое ПО, которое является частью поставляемого продукта и подвергается изменениям, вносимым в него командой разработчиков.

Таблица С.1.
Виды собираемой статистики

Виды собираемой статистики	Определение
Общее число SLOC	$SLOC_T$ = общий размер в SLOC
Конфигурированные SLOC	$SLOC_C$ = количество SLOC в текущей базовой версии
Критичные дефекты	SCO_0 = количество SCO типа 0
Обычные дефекты	SCO_1 = количество SCO типа 1
Усовершенствования	SCO_2 = количество SCO типа 2
Новые возможности	SCO_3 = количество SCO типа 3
Общее число SCO	$N = SCO_0 + SCO_1 + SCO_2$
Открытые доработки (брак)	B = совокупное число дефектных SLOC в результате SCO_0, SCO_1 и SCO_2
Закрытые доработки (исправления)	F = совокупное число исправленных SLOC
Объем работ, связанных с доработками	E = совокупная работа по исправлению SCO_0, SCO_1 и SCO_2
Время использования	UT = число часов, в течение которых данная базовая версия ПО реально эксплуатируется

- Конфигурированные SLOC ($SLOC_C$). Этот вид статистики позволяет отслеживать переход программных компонентов из состояния совершенствующейся разработки в состояние с полностью управляемой конфигурацией. Для каждого конкретного проекта данный вид статистики предоставит информацию о прогрессе и стабильности команды разработчиков. Для проектов, в которые входит многократно используемое ПО, существенный вклад в значение $SLOC_C$, а, следовательно, и в параметры прогресса и качества, будет сделан уже на ранних стадиях.
- Дефекты (SCO_0 и SCO_1). Изменения, служащие для исправления ошибок в ПО, составляют важный вид статистики, с помощью которой можно определить надежность и совершенство базовой версии. Можно ожидать, что самое большое число ошибок будет вскрыто сразу после выхода версии, после чего это число начнет со временем снижаться по мере того, как ПО будет становиться все более совершенным.
- Усовершенствования (SCO_2). Еще один побуждающий мотив для внесения изменений в основы — усовершенствования — также является ключом для оценки качества и прогресса на пути к достижению требуемого качества. Ожидания усовершенствований обратно пропорциональны ожиданиям дефектов. Поэтому в отличие от уровня дефектов, который сначала оказывается высоким, а затем снижается, уровень усовершенствований сначала низкий (все внимание уделяется дефектам), а потом возрастает. Этот феномен хорошо

согласуется с предположением, что работу по тестированию и сопровождению выполняет команда с неизменной численностью. Он может быть выражен с помощью следующего соотношения:

Работа (по дефектам) + Работа (по усовершенствованию) = Константа

Разграничение между дефектами и усовершенствованиями в некотором смысле субъективно. Определяемые здесь метрики изменений не чувствительны ни к одному из этих типов, поскольку они зависят от их суммарного эффекта. Однако различия между дефектами и усовершенствованиями могут оказывать существенное влияние на измерение завершенности (см. раздел С.2.2).

- Новые возможности (SCO_3). Изменения типа 3 отражают изменение ожиданий заинтересованной стороны каких-либо новых возможностей и функций, выходящих за рамки имеющегося контракта. Статистика изменений типа 3 анализируется отдельно, поскольку она отражает дополнительную работу, а не доработки.

- Общее количество SCO (N). Поскольку SCO – это абстрактная единица измерения изменений, важно, чтобы она оставалась неизменной для всех областей, метрики которых будут сравниваться. Так каков же уровень, на котором изменения документируются и отслеживаются? В большинстве проектов используется совершенно произвольное определение SCO, зависящее от размера, степени влияния на отдельных сотрудников и на команды и от общей культуры ССВ. Такой произвольный подход будет работать для отдельно взятого проекта, но если в каждом проекте будут применяться разные определения, то сравнение различных проектов окажется невозможным. Вообще говоря, SCO должен иметь отношение к единственному компоненту и передаваться для выполнения одному сотруднику или руководителю команды. Для такого простого стандарта не требуется более точного определения этих составляющих. Даже не столь точно определенные составляющие будут хорошо работать, а более точное определение ничего не добавит. По мере того как автоматизированные средства будут поддерживать все больший и больший набор метрик, общая методика измерений и ее составные части будут становиться все более однородными.

- Открытые доработки (B). Теоретически любые доработки приводят к повышению качества. Доработка необходима либо для того, чтобы избавиться от «плохого» качества (SCO_0 и SCO_1), либо для того, чтобы расширить компонент, повысив тем самым эффективность затрат на протяжении всего жизненного цикла (SCO_2). Для точной оценки тенденции изменения качества доработки должны рассматриваться в контексте текущей стадии жизненного цикла. Определенное количество доработок всегда необходимо при выполнении большого проекта по созданию ПО; доработки на ранних стадиях в современной модели процесса рассматриваются как признак здорового прогресса. Затянувшиеся, поздние и нулевые

доработки вследствие отсутствия конфигурированной базовой версии являются в общем случае показателями отрицательного качества. Интерпретировать эту статистику необходимо в контексте проекта. Однако доработки должны, безусловно, достигать нулевой отметки к моменту поставки продукта. Для обеспечения устойчивого процесса сбора статистики, поддающегося автоматизации, доработки можно определить как оценочное число SLOC, необходимое для внесения изменений по каждому SCO. Абсолютная точность оценок обычно не играет важной роли. Поскольку открытые доработки отслеживаются с помощью оценочных данных, а закрытые доработки отслеживаются независимо с помощью фактических данных, значения постоянно самокорректируются и остаются непротиворечивыми.

- Закрытые доработки (F). Если статистика по дефектам позволяет оценивать объем допущенных ошибок, то статистика по их устранению определяет количество исправленных ошибок. По мере решения проблемы соответствующая оценка дефектов заменяется на реально потребовавшийся объем исправлений, которые остаются в базовой версии. Хотя действительное количество исправленных SLOC (F) никогда не будет абсолютно точным, оно оказывается достаточно точным для оценки тенденций. Поскольку термин «исправленные» может иметь несколько различных значений в зависимости от того, что было добавлено, удалено или изменено, появляется необходимость в создании непротиворечивого набора правил. Измененные SLOC будут увеличивать значения B и F и не будут влиять на $SLOC_C$. Добавленный код увеличит B, F и $SLOC_C$, хотя и в разных пропорциях. Удаление кода (встречается довольно редко) без соответствующего добавления будет повышать B и понижать $SLOC_C$. При наличии значения общего объема изменений и использовании приближенных данных для определения тенденций точность и аккуратность исходных данных оказываются относительно неважными.

- Объем работ, связанных с доработкой (E). Общая работа, затраченная на выполнение SCO, является еще одной характеристикой, необходимой для определения сложности доработок. Эта деятельность должна ограничиваться техническими требованиями, разработкой ПО, созданием архитектуры, реализацией и функциональным тестированием. Разработка, управление проектом, контроль за конфигурацией, верификационное тестирование и системное тестирование для систем более высокого уровня должны быть исключены, поскольку эти виды деятельности являются скорее функцией компании, заказчика или атрибутами проекта, не зависящими от качества. Основная цель — вывести чрезвычайно изменчивую бюрократическую деятельность за рамки метрик.

- Время использования (UT). Этот важный вид статистики соответствует тому количеству часов, которое базовая версия ПО проработала в режиме реальной эксплуатации. Для некоторых систем такой вид статистики собирается с помощью непосредственного измерения времени; для других автоматизированные тесты могут эмулировать использование системы в течение дня с помощью часового теста. Например, для большинства систем обработки транзакций существует средняя ожидаемая нагрузка за день. Если эта средняя нагрузка может быть оформлена в виде некоторого сценария и выполнена данной версией ПО за один час, то это время засчитывается за 24 часа использования. В качестве другóго примера представим себе некий инструмент для разработки, используемый человеком и рассчитанный на скорость работы человека — несколько нажатий клавиш в секунду. Если автоматизированные инструменты тестирования GUI в состоянии поддерживать сценарии взаимодействия, которые способны производить тестирование продукта со скоростью в десять раз большей, в этом случае каждый час тестирования засчитывается за десять часов времени использования. Определение соответствия времени тестирования времени использования обычно не вызывает затруднений. Все это является также отличным упражнением по анализу требований, которое зачастую позволяет вскрыть факты различного понимания сценария использования разными заинтересованными сторонами.

C.2.2 Метрики качества конечного продукта

Метрики качества конечного продукта (см. таблицу C.2) показывают, какова сопровождаемость продукта по отношению к SCO типов 0, 1 и 2. SCO типа 3 не входят в это число, поскольку они переопределяют изначальное целевое качество системы, и обычно для них необходимо больше глобальных системных и программных разработок, а также некоторое переопределение требований уровня системы. Работа с этими типами изменений ведется совершенно разными способами для разных заказчиков в различных проектах, так что они лишь усложняют сравнимость данных.

Следующие метрики чрезвычайно полезны при определении и планировании объемов работ, необходимых для реализации SCO типа 3. Они оказываются также полезными, когда используются для частей продукта, например компонентов или версий. Термин «продукт» применяется для обозначения того, что подвергается измерению.

- Коэффициент брака. Этот параметр позволяет получить значение, которое можно сравнить с предыдущими проектами, с предстоящими шагами, с будущими проектами. Он определяет процент продукта, который пришлось переработать на протяжении всего жизненного цикла.

Таблица C.2.
Метрики качества конечного продукта

Метрика	Определение
Коэффициент брака	B/$SLOC_T$, процент дефектности продукта
Коэффициент доработок	E/Работа $_{по\ разработке}$, процент доработок
Коэффициент дефектности	B/N, среднее количество дефектов на один SCO
Адаптируемость	E/N, средний объем работы на один SCO
Завершенность	$UT/(SCO_0 + SCO_1)$, среднее время между двумя дефектами
Сопровождаемость	(Коэффициент брака)/(Коэффициент доработок), продуктивность сопровождения

- Коэффициент доработок. Значением этого параметра является процент работы, затраченной на переделки, по сравнению с общим объемом работ. Вероятно, это самый лучший показатель продуктивности доработок (или сопровождаемости).
- Коэффициент дефектности. Это значение определяет среднее количество дефектных и переделанных SLOC на один SCO, которое отражает способность локализации влияния изменения, присущую интегрированному продукту. ССВ должен убедиться в том, что каждый SCO пишется для единственного изменения в исходном продукте и что все они реализуются согласованным образом.
- Адаптируемость. Это значение позволяет определить ту легкость, с которой в продукт могут вноситься изменения. Хотя малое число изменений, вообще говоря, является хорошим показателем качества процесса, объем работы, затрачиваемой на внесение одного изменения, обычно оказывается более важным.
- Завершенность. Это значение является показателем текущего среднего времени наработки на отказ (MTBF) продукта. Хотя безусловной целью для завершенности всегда является бесконечность (т.е. нулевое количество дефектов), каждый проект должен довольствоваться меньшим. После передачи проекта тем, кто будет его использовать, MTBF обычно фиксируется и стабилизируется. Однако на протяжении жизненного цикла разработки действия по сопровождению должны, по идее, увеличивать завершенность каждой конкретной версии, а тенденции, определяемые на основании нескольких версий, должны демонстрировать улучшения, направленные на достижение цели всего проекта в смысле завершенности.
- Сопровождаемость. Это значение определяет соотношение между стоимостью сопровождения и стоимостью разработки. Оно обеспечивает приведение к единому знаменателю для сравнения между собой различных проектов. Поскольку числитель в формуле сопровождаемости выражается в терминах объема работы, а знаменатель — в терминах SLOC, то это — степень продуктивности (единица

работы, отнесенная к SLOC). Простые математические преобразования показывают, что сопровождаемость (или качество сопровождения — Q_M) эквивалентно следующему:

$$Q_M = \text{Продуктивность}_{\text{Сопровождения}}/\text{Продуктивность}_{\text{Разработки}}$$

Например, если (Коэффициент брака) = (Коэффициент доработок), то продуктивность внесения изменений равна продуктивности разработки и $Q_M = 1$. Интуитивно понятно, что значение 1 соответствует «слабому» уровню сопровождаемости, поскольку оказывается проще внести изменения в уже существующее ПО, чем разрабатывать альтернативное с самого начала. Тот факт, что в традиционных проектах обычно тратится $2 на сопровождение на каждый доллар, затраченный на разработку [Boehm, 1987], может служить в качестве отправной точки при определении того, что является «хорошим» уровнем сопровождаемости. Представим себе некоторую область деятельности, связанную с ПО, со средним временем жизни продукта 16 лет и со средними ежегодными дефектами в размере 12%. Если $Q_M = 1$, то соотношение между затратами на сопровождение и затратами на разработку окажется равным приблизительно 1:2, другими словами, сопровождаемость окажется такой, которая принимается (грубо) за норму в индустрии ПО. Значение сопровождаемости намного меньшее единицы в большинстве случаев означает, что сопровождаемость продукта очень высока, по крайней мере, по отношению к затратам на разработку и к опыту традиционной разработки.

Приведенное выше описание определяет идеализированные тенденции для рассматриваемых параметров. Реальные ситуации, складывающиеся в проекте, никогда не бывают идеальными. Все заинтересованные стороны должны понимать ту степень, в которой эти параметры отклоняются от идеала. Эти параметры полезно использовать для каждой ступени развития проекта в целом, а также для сравнения с другими проектами.

C.2.3 Внутренние показатели прогресса

Определения внутренних показателей прогресса даны в таблице C.3. Ожидаемые тенденции обсуждаются ниже и демонстрируются на рис. C.1 и C.2.

- Стабильность доработок. Эта метрика является количественным выражением разницы между общим числом доработок и закрытыми доработками. Ее важность заключается в том, что она показывает, возрастает ли уровень исправлений по мере увеличения уровня брака. На рис. C.1 дается пример благополучного проекта, в котором уровень доработок не расходится с уровнем брака (за исключением коротких периодов времени). Уровень брака должен отслеживаться и по отношению к уровню $SLOC_C$ в поставляемом продукте, поскольку уровень работ, направленных на тестирование и сопровождение, меняется на протяжении жизненного цикла. Для этого необходима следующая метрика.

Таблица С.3.
Определения внутренних показателей прогресса

Показатель	Определение
Стабильность доработок	B-F, брак минус исправленные ошибки в зависимости от времени
Накопившиеся доработки	(B-F)/$SLOC_C$, доработки, открытые на данный момент
Тенденция изменения коэффициента дефектности	Изменение коэффициента дефектности в зависимости от времени
Тенденция изменения адаптируемости	Изменение адаптируемости в зависимости от времени
Тенденция изменения завершенности	Изменение завершенности в зависимости от времени

- Накопившиеся доработки. Накопившиеся доработки — это нуждающийся в доработке процент от объема существующей на данный момент базовой версии продукта, выраженного в $SLOC_C$. В общем случае объем накопившихся доработок будет расти до некоторого приемлемого уровня после завершения создания начальной версии, поскольку тестирование позволит выявить необходимые изменения. Уровень накопившихся доработок должен оставаться относительно стабильным на протяжении выполнения программы тестирования, после чего упасть до нуля. Большой объем изменений или непрерывное возрастание числа накопившихся доработок требует внимательного изучения. Непрерывный рост может свидетельствовать о нестабильности и отклонении от плана.
- Тенденция изменения коэффициента дефектности. Изменение этого значения показывает, как степень изменений эволюционирует на протяжении жизненного цикла проекта. Общая тенденция позволяет заглянуть внутрь качества (насколько хорошо архитектура приспособлена к локализации изменений) и управления (соответствие графику работ и последовательное изменение рисков). Большая часть тривиальных ошибок отлавливается и исправляется в процессе тестирования отдельных компонентов. Это значение относится к нетривиальным ошибкам, вкравшимся в базовую конфигурацию. Хотя дать количественное определение того, что именно может считаться хорошей тенденцией, довольно трудно, следующее эмпирическое правило выполняется для успешных проектов: среднее количество SCO должно быть эквивалентно размеру отдельного модуля программы (самый низкий уровень независимо компилируемых элементов программы). Например, средний дефект, приходящийся на один SCO для ПО, написанного на языке C++ (в котором размер среднего элемента программы равен приблизительно 50 строк кода), должен составлять по окончании проекта приблизительно 50 строк. Для зрелого процесса итерационной разработки ошибки, допущенные на ранних стадиях (ошибки при разработке,

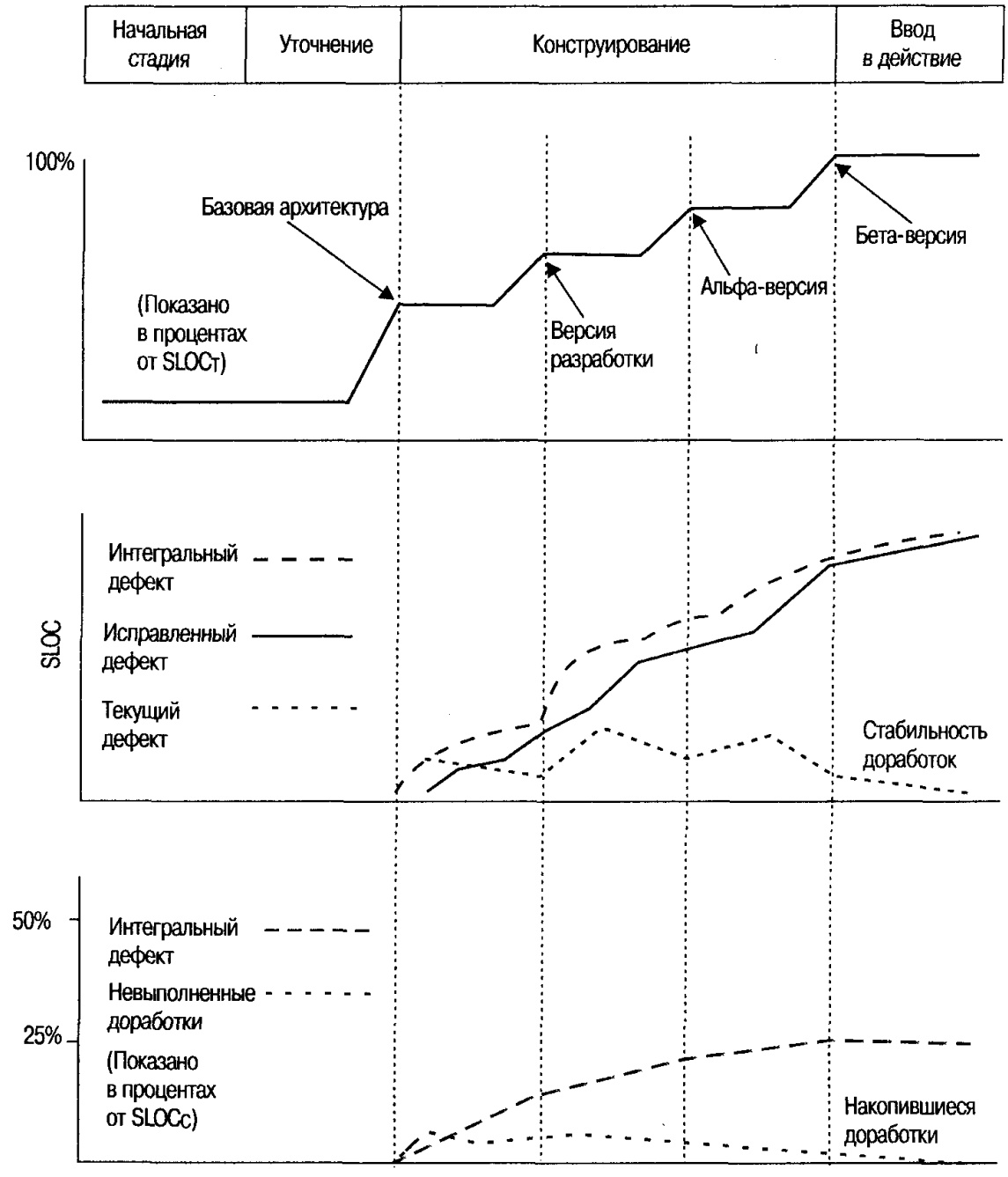


Рис. С.1. *Ожидаемые тенденции изменения внутренних индикаторов прогресса*

которые касаются многих компонентов и различных исполнителей), потребуют, скорее всего, большего количества доработок, чем ошибки, допущенные на более поздних стадиях (ошибки при реализации, чье влияние ограничивается одним компонентом или одним человеком). Тенденция коэффициента дефектности к возрастанию с течением времени однозначно свидетельствует о том, что архитектура проекта приходит в упадок.

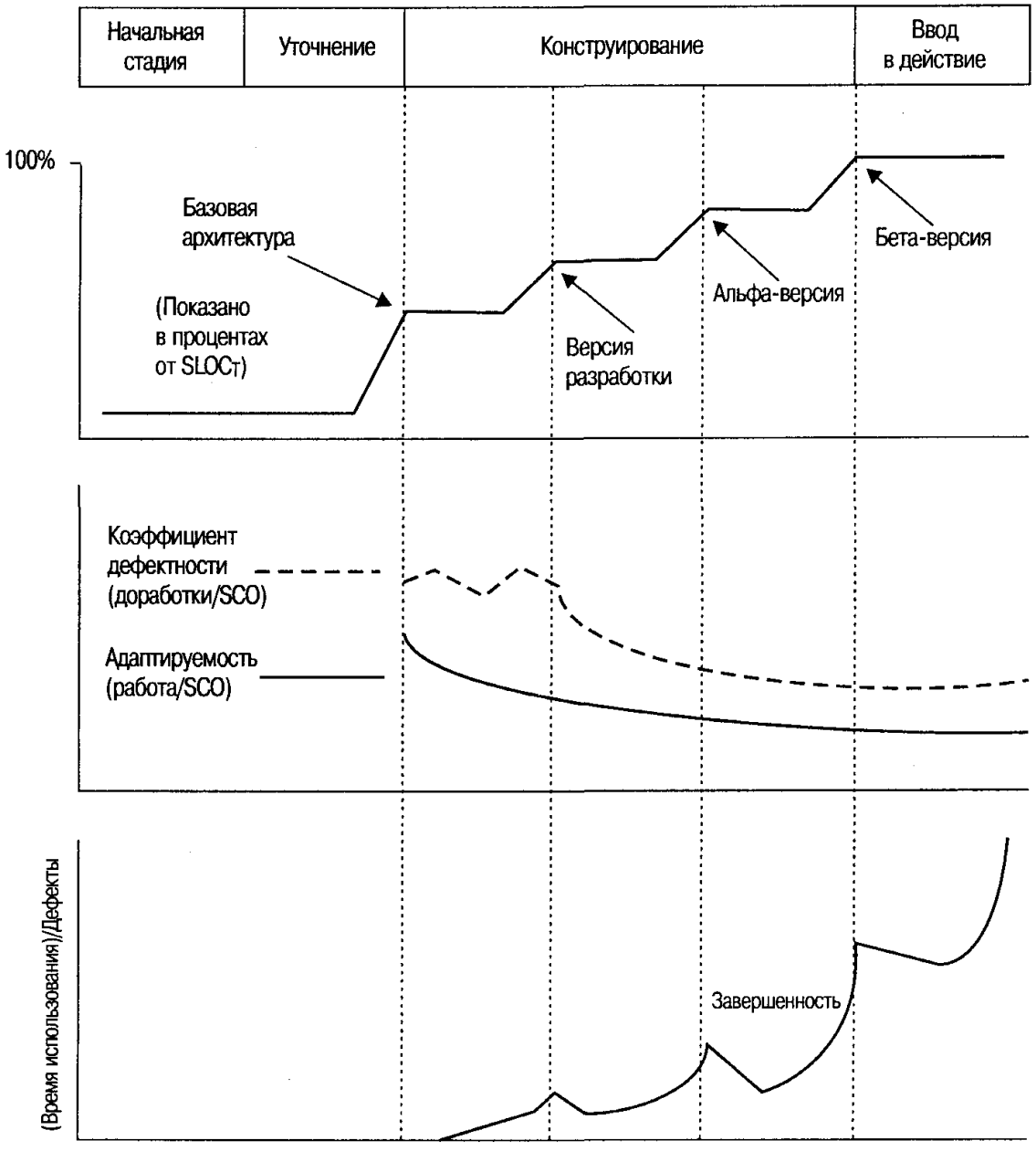


Рис. С.2. *Ожидаемые тенденции изменения качества*

- Тенденция изменения адаптируемости. Это значение предоставляет механизм для оценки тенденций, которые присущи сложности изменений, в отличие от степени изменений. Если процедура внесения изменений оказывается простой, то число изменений в рамках проекта, скорее всего, будет возрастать, улучшая тем самым его качество. В случае традиционного процесса внесение изменений на поздних этапах жизненного цикла обходится дороже. Целью современного итерационного процесса является достижение такой стройности процесса и архитектуры, которая бы упростила внесение изменений — с более предсказуемым результатом — на поздних этапах жизненного цикла. Доработки обычно стабилизируются, а

не упрощаются с течением времени. Однако именно в этом и заключается принципиальное отличие от традиционного процесса. Хорошую тенденцию трудно выразить количественно в абсолютных значениях. На практике для успешных проектов средние затраты на внесение одного изменения должны составлять менее одной человеко-недели.

- Тенденция изменения завершенности. Ожидаемое изменение этого значения просто объяснить для одной версии. Однако современные проекты по созданию ПО состоят из нескольких итераций и этапов, работы по которым и сроки выполнения которых частично перекрываются. Оценка завершенности системы в целом оказывается более сложной задачей, чем оценка завершенности одной версии. В случае отдельной версии можно ожидать получения относительно незавершенного продукта (с часто встречающимися дефектами), для которого характерен быстрый рост завершенности по мере внесения исправлений в процессе сопровождения. Ожидаемое поведение этого значения в случае простого проекта, приведенное на рис. С.2, таково, что для последовательных версий базовая версия в целом будет содержать меньшее количество дефектов, а время использования будет расти. Соответственно, все заметнее и заметнее будет становиться рост надежности. Экспоненциальный рост, показанный на рисунке, для большинства систем, вероятнее всего, нереален. Более реалистичным является рост линейный. Пройденный путь может быть различным, но для благополучных процесса и архитектуры не должно существовать продолжительных понижений завершенности, а кратковременные понижения могут не иметь видимых причин.

С.3 ИСПОЛЬЗОВАНИЕ МЕТРИК ИЗМЕНЕНИЙ НА ПРАКТИКЕ

В разделе 13.1 описаны некоторые цели, присущие правильной программе применения метрик. Эти цели повторяются ниже вместе с обсуждением того, насколько рассматриваемые здесь метрики соответствуют целям.

- **Метрики должны быть простыми, объективными, их должно быть легко собирать, легко интерпретировать и трудно интерпретировать неправильно**. Количество видов статистики, которые должны собираться по базе данных, содержащей SCO, невелико: менее десяти. Они должны легко подсчитываться и определяться, хотя на практике многие элементы этих подсчетов оказываются неоднозначными. В зависимости от дисциплины, непротиворечивости и уровня автоматизации, присущих процессу, используемому в данной организации, определение и сбор данных для этих метрик может быть относительно простым. С другой стороны, конкретной организации, выполняющей разнообразные проекты по созданию ПО, возможно, будет довольно сложно определиться с приемлемой

практикой. Различные точки зрения, обеспечиваемые этими метриками, в большинстве случаев могут быть легко интерпретированы. Большинство тенденций являются очевидно хорошими или очевидно плохими. Многие данные зависят от контекста, но, располагая данными для множества проектов, собранными по одинаковой методике, можно без труда судить о сходстве и различиях между ними.

- **Сбор метрик должен быть автоматизированным и ненавязчивым, т.е. не должен пересекаться с выполнением разработчиками своих задач.** Весь сбор данных и их анализ, необходимый для данного подхода к метрикам, может быть — и был — автоматизирован. В то время как разработчики создают рабочие продукты, система контроля за конфигурацией может использоваться в качестве инструмента для сбора и обработки всех данных, необходимых для определения метрик и тенденций их изменения.

- **Метрики позволяют получать непротиворечивые оценки на протяжении всего жизненного цикла, особенно на ранних его стадиях, когда работы, направленные на повышение качества, требуют больших затрат.** Подход, описываемый в данной книге, определяется с точки зрения сопровождения ПО. Однако итерационный процесс можно рассматривать как объединение деятельности по разработке и сопровождению в общий набор видов деятельности на протяжении жизненного цикла, который использует те же приемы и инструменты. С этой точки зрения итерационный подход можно рассматривать как ускорение создания базовой версии с тем, чтобы изменения, вносимые в эту версию, ее дальнейшее развитие и вопросы качества могли быть применены для улучшения инструментального оснащения процесса. Для традиционной технологии это был бы чреватый ошибками вид деятельности, выполняемый вручную. При современном развитом автоматизированном управлении изменениями и поддержке «круговой» разработки возможности по внесению изменений существенно возросли, а переход к итерационному процессу стал технически осуществим и экономически выгоден.

- **Метрики и тенденции их изменения нужны как управленческому, так и инженерному персоналу для обмена информацией о прогрессе и качестве, представленной в согласованном формате.** Эти метрики связаны с реальными измерениями рабочих продуктов разработки ПО. Они извлекаются непосредственно из изменяющейся базовой версии продукта, а не из какой-либо документации или субъективных суждений. Разработчики ПО должны принять и использовать эти объективные метрики, чтобы избежать плохих технических и управленческих решений. Представленные метрики просты: их понимание доступно большинству заинтересованных сторон, они поддаются автоматизации и при разумном использовании позволяют проводить сравнение с метриками других проектов.

Приложение D

Практический пример: CCPDS-R

В этом приложении представлен для подробного изучения практический пример успешного проекта по созданию ПО, в котором использовались многие из методов, описанных в книге. Термин «успешный» означает выполнение проекта в рамках бюджета, в срок и в приемлемом для заказчика виде. Проект новой системы обработки и вывода данных для командного центра (Command Centre Processing and Display System-Replacement, CCPDS-R) осуществлялся по заказу военно-воздушных сил США компанией TRW Space and Defence, расположенной в Редондо Бич, штат Калифорния. В целом проект включал в себя разработку систем, обеспечение аппаратурой и создание ПО, причем на каждое из этих трех основных направлений деятельности направлялось примерно по одной трети средств от общей суммы. Сроки разработки охватывали период с 1987 по 1994 г.

Ключевые моменты

- Объективный практический пример является верным показателем зрелости организации и зрелости процесса выполнения проекта. Индустрии ПО требуется наличие как можно большего количества таких практических примеров, как CCPDS-R.
- Все значения метрик получались непосредственно из рабочих продуктов проекта. Эти данные использовались для управления проектом и были приняты разработчиками, менеджерами и заинтересованными сторонами.
- Проект CCPDS-R явился одним из первых новаторских проектов, в котором были опробованы многие современные подходы к управлению.
- В этом приложении дается практический контекст для тех методов, дисциплин и мнений, которые содержатся в данной книге.

Задача по созданию ПО состояла в разработке трех различных программных систем с общим объемом более одного миллиона строк исходного кода. В данном практическом примере основное внимание уделяется программной системе, разрабатывавшейся в первую очередь. Она называлась Подсистемой общего назначения, для ее создания потребовалось написать около 355 000 строк исходного кода. В результате создания

Подсистемы общего назначения были также получены повторно используемая архитектура, зрелый процесс и интегрированная среда для последующей эффективной разработки двух других программных подсистем приблизительно такого же размера. Следовательно, предлагаемый практический пример представляет примерно одну шестую часть от всех работ по проекту CCPDS-R.

Этот практический пример не совпадает в точности ни с процессом управления, представленным в настоящей книге, ни с одной из применяемых в настоящее время технологий, однако в нем были использованы в основном те же методы, и руководствовался он тем же духом и теми же приоритетами. Компания TRW выполнила проект, уложившись в рамки бюджета и в сроки, а пользователи получили даже больше, чем ожидали. Компания TRW была награждена премией за отличное качество в области ракетных и космических систем оповещения в 1991 г. «за постоянную, продолжительную работу в области разработки систем и выполнения проектов». Проект, подобный CCPDS-R, в наши дни мог бы быть разработан более эффективно. При использовании современных технологий и усовершенствованных процессов, среды и уровня автоматизации в современных условиях этот проект, вероятно, был бы реализован за вдвое меньшее количество времени и за одну четвертую стоимости при неизменном качестве.

D.1 ОБЩИЕ ПОЛОЖЕНИЯ ДЛЯ ДАННОГО ПРАКТИЧЕСКОГО ПРИМЕРА

Я проработал над проектом CCPDS-R в течение шести лет, поэтому данное приложение написано от лица непосредственного участника. В мои задачи входило управление разработкой основных технологий, создание технических и финансовых предложений, проведение практической разработки ПО и управление созданием ПО на ранних этапах реализации общих функциональных возможностей.

Я попытался представить точную картину проекта CCPDS-R. Все данные являются в основном историческим фактом, а вот субъективные комментарии и оценки значимости принадлежат мне. В качестве источников данных использовались опубликованные статьи, руководства для внутреннего пользования компании TRW, поставляемая в соответствии с контрактом документация — все это можно было извлечь из реальных рабочих продуктов по проекту CCPDS-R — и мой собственный практический опыт. В некоторых случаях я редактировал данные, чтобы избавиться от ненужной точности и исключить противоречия в исходных документах, которые разрабатывались на различных стадиях жизненного цикла. Моей целью являлось создание относительно непротиворечивого описания, поэтому мне пришлось исключить некоторые детали, для которых потребовались бы не относящиеся к предмету пояснения.

В индустрии ПО существует много успешных проектов (недостаточно, но много), однако хорошие практические примеры отсутствуют. Имеется небольшое количество тщательно документированных проектов с объективным описанием того, что было сделано, чего не было сделано и

почему. Это являлось одним из главных моих побудительных мотивов для той степени подробности, с которой написано приложение. В него включены специфичные подробности, подходы и результаты по трем причинам:

1. Написание этого практического примера не потребовало большого труда. Проект CCPDS-R уникален в своем подходе к тщательности и автоматизации работы с метриками. Все данные были получены непосредственно из реальных «исторических» рабочих продуктов, созданных в процессе выполнения проекта.
2. Данная разновидность объективных практических примеров является точным индикатором зрелости организации и зрелости процесса выполнения проекта. Абсолютные «исторические» цифры могут иметь весьма ограниченное применение. Однако тенденции, извлеченные уроки и относительная важность приоритетов являются отличительными особенностями любой успешной разработки ПО.
3. В предшествующих главах многие технические и управленческие подходы обсуждались в общем. Данное приложение предоставляет по крайней мере одну практическую точку отсчета для оценки работы.

Мои комментарии, касающиеся значимости методов, дисциплин и мнений, обсуждавшихся в предыдущих главах, приводятся в рамках на сером фоне.

D.2 ОБЩИЙ ОБЗОР ПОДСИСТЕМ

В результате осуществления проекта CCPDS-R была создана крупномасштабная высоконадежная система командования и контроля, которая обеспечивала Национальный Командный Центр информацией о запусках ракет. В качестве посредника выступало подразделение электронных систем Главного штаба военно-воздушных сил, расположенного на военно-воздушной базе в Хэнскоме, штат Массачусетс. Главным пользователем являлось космическое командование США, а контракт на полномасштабную разработку был заключен с TRW's Systems Integration Group в 1987 г. Контракт по созданию CCPDS-R предусматривал разработку трех подсистем:

1. Подсистема общего назначения являлась главной системой оповещения о запусках ракет в рамках программы Cheyenne Mountain Upgrade. Для ее создания потребовалось написать около 355 000 строк исходного кода, срок ее разработки составлял 48 месяцев, и она являлась основой для подсистем, которые разрабатывались позже (повторно используемые компоненты, инструментарий, среда, процесс, процедуры). Подсистема общего назначения была рассчитана на основную установку в Cheyenne Mountain, дублирующая подсистема устанавливалась на базе военно-воздушных сил в Оффутте, штат Небраска.

2. Подсистема обработки и вывода (Processing and Display Subsystem, PDS) была уменьшенным вариантом системы вывода оповещения о запусках ракет, которая предназначалась для всех командующих, ответственных за пуск ядерных ракет. ПО PDS (около 250 000 SLOC) устанавливалось на удаленных, работающих в режиме «только чтение» рабочих станциях, разбросанных по всему миру.
3. Подсистема STRATCOM (около 450 000 SLOC) обеспечивала как оповещение о запусках ракет, так и возможность принудительного управления для запасного центра оповещения о запусках ракет в командном центре Стратегического командования.

Общий процесс создания ПО

Создание ПО в рамках проекта CCPDS-R состояло из двух отдельных стадий: определения концепции (ОК) и полномасштабной разработки (ПМР). На контракт на выполнение стадии ОК претендовали пять компаний, были заключены два контракта с фиксированной суммой в $2 млн. каждый. Фирмы, получившие контракты, должны были вложить также собственные ресурсы по своему усмотрению на то, чтобы определиться с самым выгодным предложением для стадии ПМР. На рис. D.1 показаны общий процесс создания и продукты, полученные на каждой стадии.

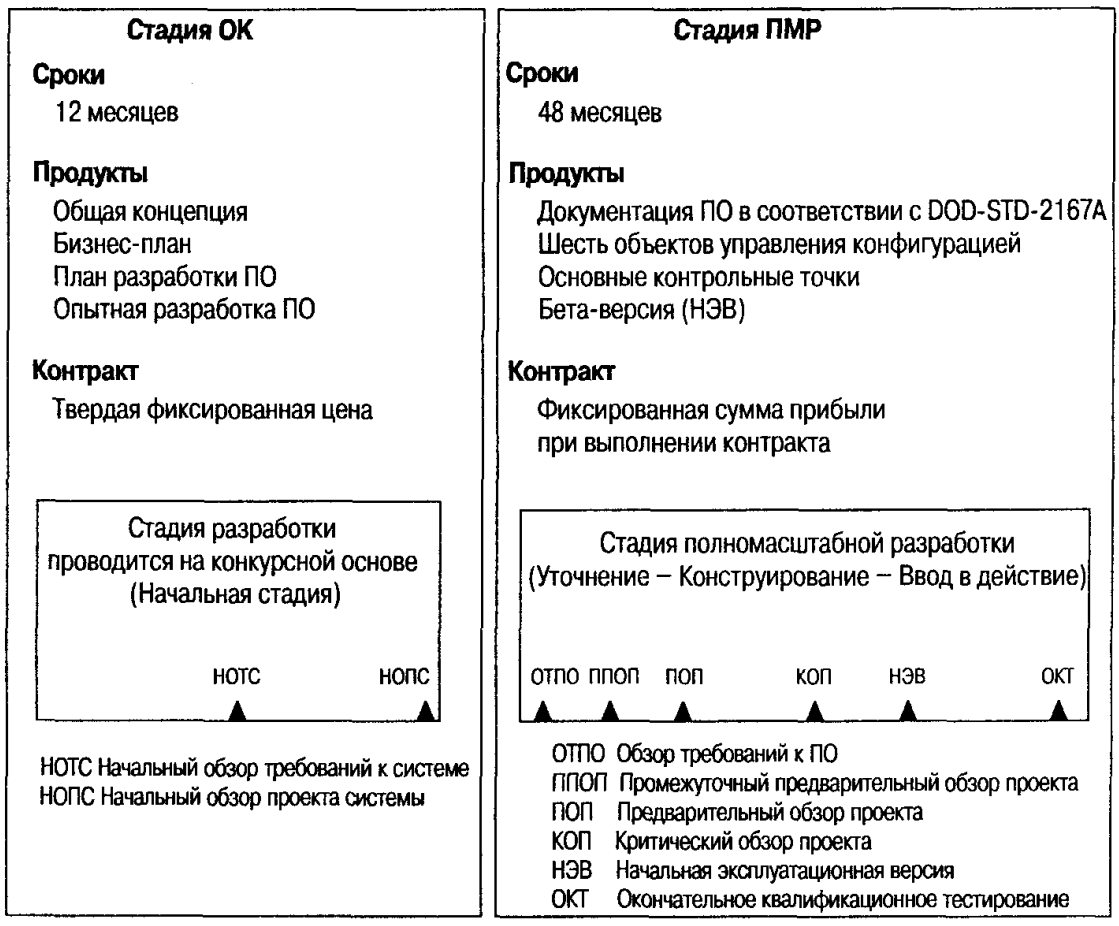


Рис. D.1. *Обзор жизненного цикла проекта CCPDS-R*

Стадия ОК по своим задачам близка к начальной стадии. Ее главными продуктами являются спецификации системы (документ с общей концепцией), предложение по стадии ПМР (бизнес-план, включающий в себя описание технического подхода, фиксированную сумму прибыли и предложение по общей стоимости контракта) и план разработки ПО. Стадия ОК включает в себя также обзор проекта системы, технические совещания для обмена информацией с заинтересованными сторонами от правительства (заказчик и пользователь) и некоторые оговоренные в контракте документы. Эти действия и продукты позволяют произвести исходную оценку ПМР, основанную как на работе, продемонстрированной предложенной подрядчиком командой, так и на предложениях по ПМР.

С точки зрения ПО появляется еще один дополнительный критерий для исходного выбора, который включен в работы по подготовке предложений по ПМР, — опытная разработка ПО. Это был уникальный, но эффективный подход для оценки возможностей двух конкурирующих между собой подрядчиков по осуществлению разработки ПО. Военно-воздушные силы чрезвычайно интересовал общий риск данного проекта: от предыдущих проектов сложилась весьма унылая картина. Ответственные за закупку ПО в военно-воздушных силах испытывали также сильное разочарование в таких ситуациях, когда первоклассная команда по разработке предложений оказывалась вовсе не той командой, которой предстояло выполнять контракт после его заключения, а предложения подрядчика обычно приукрашивали его подходы или возможности по сравнению с тем, что он мог реально осуществить.

Проект CCPDS-R включал в себя также большую составляющую по разработке ПО и являлся одним из первых проектов, в котором использовался язык программирования Ada. В тот момент существовали серьезные опасения относительно того, что среда разработки Ada, процесс разработки подрядчика и программы подрядчика по обучению могут оказаться недостаточно зрелыми для использования в крупномасштабных проектах. Целью опытной разработки ПО и являлась демонстрация того, что предлагаемый подрядчиком процесс создания ПО, среда Ada и команда разработчиков имеются в наличии, являются зрелыми и могут быть представлены заказчику.

Опытная разработка ПО началась немедленно после рассмотрения предложений по ПМР. Заказчик предоставил обоим соискателям спецификации несложного «эмулятора для оповещения о запуске ракет» на двух страницах. Этот эмулятор должен был удовлетворять некоторым фундаментальным требованиям из числа предъявляемых к полномасштабной системе CCPDS-R, включая распределенную архитектуру, гибкий пользовательский интерфейс и основные сценарии обработки потока событий CCPDS-R, связанного с оповещением о запуске ракет. Требования к опытной разработке включали в себя следующее:

- Использование предложенной команды разработчиков ПО
- Применение для разработки ПО предложенных методов и инструментов

- Использование содержащегося в предложениях по ПМР плана разработки ПО
- Предоставление заказчику макетного обзора проекта через 23 дня после получения спецификаций

Опытная разработка ПО должна была дать объективную характеристику пригодности предложенных каждым из подрядчиков подходов к разработке ПО.

Результаты, полученные командой по разработке проекта CCPDS-R в компании TRW, оказались впечатляющими. Они продемонстрировали заказчику, что команда готова, работоспособна и компетентна для проведения в жизнь предложенного подхода к созданию ПО. На эту работу было затрачено приблизительно 12 человеко-месяцев (12 человек на полный рабочий день в течение 23 дней).

Был создан подробный план, в который вошли расписание работ, распределение ответственности и ожидаемые результаты, что позволяло отслеживать прогресс. План включал в себя две итерации по созданию архитектуры и все контрольные точки и рабочие продукты, предложенные в плане проекта. Опытная разработка дала следующие результаты:

- Были созданы и продемонстрированы четыре основных варианта использования.
- Была разработана скелетная архитектура ПО, создан ее прототип и подготовлена документация, причем в прототип вошли два выполняемых распределенных процесса, пять параллельно выполняющихся задач (независимые потоки управления), 8 компонентов, 72 интерфейса между компонентами.
- Было разработано и написано 4163 исходных строк для прототипов компонентов. В демонстрационную версию было включено также несколько повторно используемых компонентов общим объемом в несколько тысяч строк.
- Были пройдены три контрольные точки и разрешено более 30 рабочих проблем.
- Создание 11 документов (в соответствии с предложенными рабочими продуктами) продемонстрировало уровень автоматизации, присущий средствам документирования.
- Были использованы средства Digital Equipment Corporation VAX/VMS, среда Rational R1000, шаблоны документов LaTeX и несколько специально созданных инструментов.
- Были определены некоторые усовершенствования процесса и инструментов, которые необходимо произвести. Концепция изменения плана, требований, процесса, проекта и среды в каждой основной контрольной точке была признана потенциально рискованной, тем не менее была реализована посредством строгого управления изменениями.

Как оказалось, опытная разработка стала определяющим фактором при выборе подрядчика на выполнение контракта по созданию CCPDS-R. Компания TRW предложила основанный на демонстрациях подход с упреждающей разработкой архитектуры, а также сумела с успехом представить свою рабочую концепцию в реальных условиях, хотя и в малом масштабе и в течение весьма короткого промежутка времени. Несмотря на то, что предложение компании TRW было более чем на 20% дороже предложения ее конкурента, был выбран подход компании TRW как лучший и сопровождающийся наименьшим риском. Заключение контракта с компанией TRW произошло во многом благодаря тому, что удалось успешно выполнить опытную разработку ПО, а также тому, что компания TRW оказалась способна продемонстрировать заслуживающий большего доверия и менее рискованный процесс в реальных условиях.

> Опытная разработка ПО служила тем же целям, что и оценка возможностей по созданию ПО (SEI Software Capability Evaluation), рассмотренная в приложении E. В предложении каждого из конкурентов содержался план разработки ПО — та часть организации процесса, в которой расписывалось, что надо делать. Опытная разработка продемонстрировала, что предлагаемая организация в состоянии работать так, как об этом объявлено.

D.3 ОРГАНИЗАЦИЯ ПРОЕКТА

При подготовке к выполнению проекта CCPDS-R компания TRW уделила повышенное внимание формированию хорошей команды. Команда, трудившаяся в течение стадии OK, представляла собой основу команды по созданию архитектуры (см. раздел 11.2), отвечавшей за эффективность стадии разработки. На эту команду возлагалась ответственность за следующие основные действия:

- Проанализировать и определить требования проекта
- Определить и разработать архитектуру самого верхнего уровня
- Распланировать различные виды деятельности по разработке ПО в течение стадии ПМР
- Сконфигурировать процесс и среду разработки
- Установить доверительные и направленные на достижение успеха отношения между заинтересованными сторонами

Команда на стадии OK была небольшой и высококвалифицированной, а иерархия внутри команды была малозаметной, если вообще была. Одним из ее исключительных атрибутов явился полный набор талантов. Были представлены все необходимые квалификации, а конкуренция между сотрудниками практически полностью отсутствовала.

Команда для стадии ПМР была сформирована путем перевода многих членов команды, работавшей на стадии OK, на руководящие должности и

увеличения числа сотрудников до уровня, необходимого для проведения полномасштабной разработки. На рис. D.2 показаны эволюция организации создания ПО и распределение ответственности в процессе ПМР.

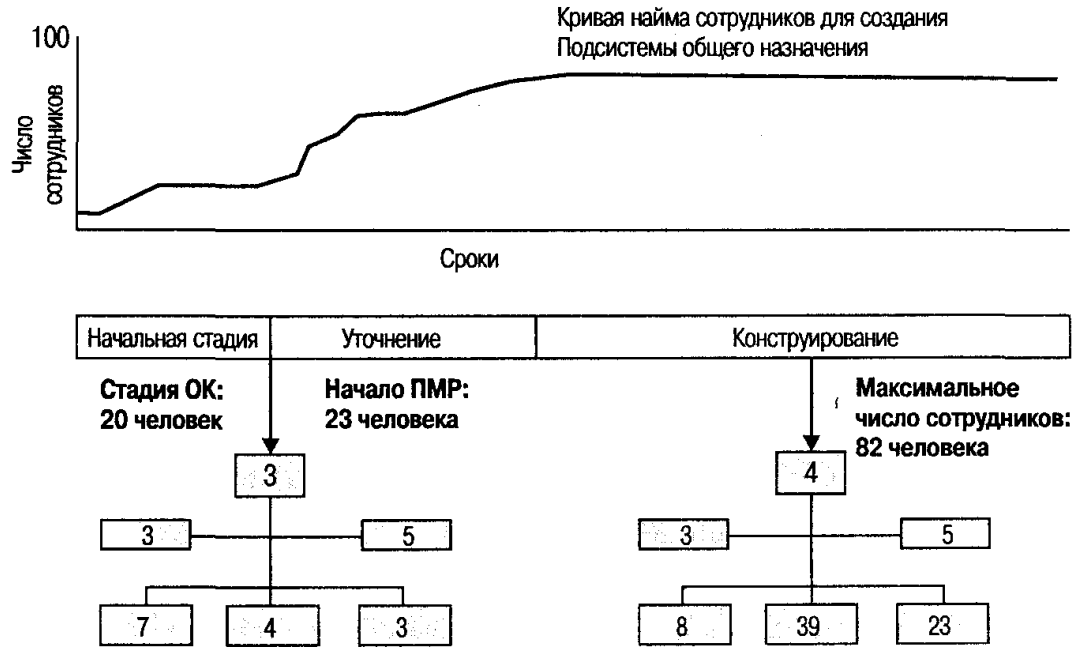
> Организационная структура и распределение обязанностей для проекта CCPDS-R похожи на те, что были рекомендованы на рис. 11.2. Изменения количества сотрудников соответствуют данным, приведенным в таблице 10.2.

D.4 ОБЗОР ПОДСИСТЕМЫ ОБЩЕГО НАЗНАЧЕНИЯ

ПО, называемое Подсистемой общего назначения, охватывает шесть объектов управления конфигурацией (computer software configuration item, CSCI). (CSCI является жаргоном заказчиков из правительства и употребляется для обозначения набора компонентов, которые управляются, конфигурируются и документируются как единое целое и разработка которых поручается одной команде разработчиков.) CSCI определены и описаны в стандарте Министерства обороны DOD-STD-2167A [DOD, 1988]. К CSCI относятся следующие объекты:

1. Служба архитектуры сети (CAC). Это основное промежуточное ПО состоит из повторно используемых компонентов для управления сетью, организации взаимодействия между процессами, инициализации, реконфигурации, управления в аномальных ситуациях и инструментов, обеспечивающих нормальное состояние и производительность ПО. Этот CSCI разрабатывался таким образом, чтобы его можно было использовать во всех трех подсистемах CCPDS-R.
2. Системные службы (CC). Охватывает скелетную архитектуру, распределение данных в режиме реального времени, глобальные типы данных и интерфейс для взаимодействия с оператором компьютерной системы.
3. Координация вывода (КВ). В этот CSCI входят управление пользовательским интерфейсом, форматы вывода и распределение вывода по дисплеям.
4. Тестирование и эмуляция (ТИЭ). Охватывает создание сценариев тестирования, ввод тестовых сообщений, сохранение результатов тестирования и выполнение сценариев.
5. Основная обработка данных (ООД). Охватывает различные алгоритмы оповещения о запуске ракет по ранним сообщениям, получаемым от радаров, ядерной детонации и со спутников.
6. Внешнее взаимодействие (ВВД). Охватывает внешние интерфейсы взаимодействия с другими системами, а также ввод, вывод и управление протоколом.

Определяющие характеристики каждого CSCI сведены в таблицу D.1.



Организация проекта и распределение обязанностей на стадии ПМР

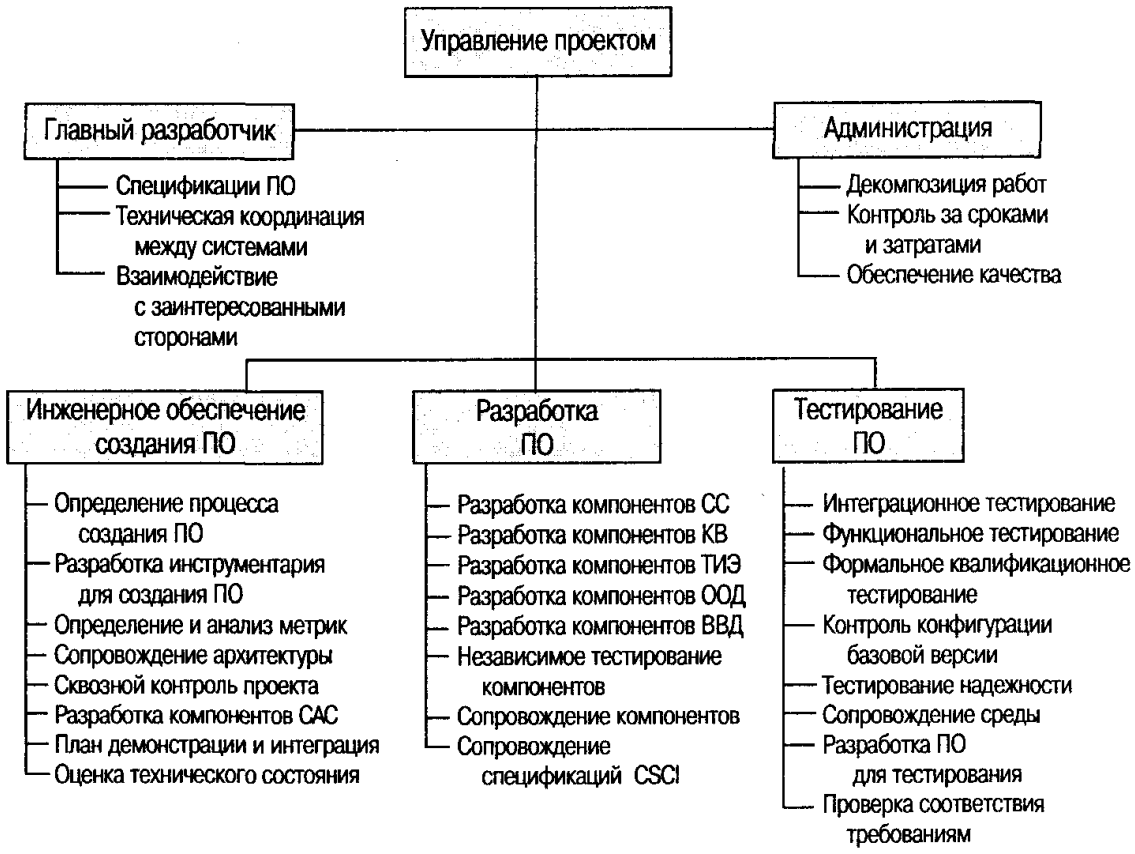


Рис. D.2. *Организация стадии полномасштабной разработки проекта*

Таблица D.1.
Итоговая таблица по CSCI

CSCI	Размер (SLOC)	Сложность	Позитивные качества (+) и нерешенные проблемы (–)
CAC	20 000	Очень высокая	+ Очень опытная команда; продукт второго поколения – Повторное использование в различных подсистемах; высокая производительность, надежность
CC	160 000	Высокая	+ Код, частично сгенерированный с помощью инструментов; стабильная CAC – Чрезвычайно большое количество глобальных интерфейсов, типов, компонентов
KB	70 000	Умеренная	+ Гибкость при разработке вывода; форматы, сгенерированные с помощью инструментов - Жесткие требования к производительности; постоянные изменения
ТИЭ	10 000	Низкая	+ Простые приложения; частичная обработка в режиме оффлайн – Сторонняя команда; ограниченные ресурсы среды
ООД	15 000	Умеренная	+ Опыт в данной области; простая обработка – Сторонняя команда; жесткие требования к производительности; чрезвычайно большое число заинтересованных сторон, вовлеченных в процесс согласования и утверждения проектных решений
ВВД	80 000	Высокая	+ Высококвалифицированные сотрудники; предшествующий опыт в данной области – Жесткие требования к производительности; неустойчивый внешний интерфейс
Итого	355 000	Высокая	Крупный масштаб, высокая производительность, высокая надежность

Скелетная архитектура ПО

Процесс создания ПО в рамках проекта CCPDS-R был адаптирован к применению языка Ada и повторно используемых компонентов промежуточного ПО для быстрого построения распределенной архитектуры. CSCI CAC содержит эти базовые компоненты, поэтому изначально он создавался на основе независимого финансирования исследований и разработки до заключения контракта по проекту CCPDS-R. Эти компоненты являлись решением по промежуточному ПО первого поколения, которое обеспечивало реальный компонентный подход к созданию распределенной архитектуры. Воплощение общих задач, процессов, межпрограммных интерфейсов и набора операций в виде выполняемой инфраструктуры называется скелетной архитектурой ПО (САПО). Работы и

демонстрации, связанные с САПО Подсистемы общего назначения, находились в центре внимания на ранних этапах. Это превосходный пример процесса с упреждающей разработкой архитектуры.

САПО представляет собой декларативную часть решения, содержащую все управляющие структуры самого верхнего уровня, интерфейсы и типы данных, обмен которыми происходит в рамках этих интерфейсов. В случае проекта CCPDS-R эта часть решения включает в себя:

- Все главные программы на языке Ada
- Все задачи, написанные на языке Ada, и их атрибуты
- Все межпрограммные интерфейсы (для асинхронного взаимодействия между задачами), атрибуты этих интерфейсов и соединения с другими интерфейсами
- Типы данных для объектов, обмен которыми производится через межпрограммные интерфейсы
- Компоненты САС для инициализации, управления состоянием процесса и задач, организации взаимодействия между процессами, обработки ошибок, отслеживания общего состояния и хода выполнения, инструментального оснащения, управления сетью, регистрации и контроля за сетью

САПО подвергается процессу компиляции, но она не будет выполнять большого количества сценариев (кроме инициализации и ожидания), пока не будет добавлено ПО для чтения сообщений, их обработки и передачи в прикладные задачи. Главной целью САПО является обеспечение структурой и системой интерфейсов, которые позволили бы подключать новые компоненты для реализации новых возможностей в различных направлениях. Существуют два важных аспекта верификации и оценки САПО: компиляция и выполнение. Создание и совместная компиляция всех объектов САПО сами по себе являются важным и нетривиальным способом оценки, который позволяет получить значимую информацию относительно непротиворечивости и качества САПО. Создание компонентов и оценка их выполнения в рамках САПО дают дополнительную информацию относительно структурной целостности и семантики.

Кроме того, САПО предоставляет информацию для обсуждения проблем интеграции и развития архитектуры. Важно создать САПО на ранних этапах и включить ее в состав стабильной базовой версии, внесение изменений в которую управляется и учитывается для получения данных относительно стабильности архитектуры. В рамках проекта CCPDS-R первая базовая версия САПО была инсталлирована (после трех неформальных итераций) приблизительно на тринадцатом месяце непосредственно перед достижением контрольной точки предварительного обзора проекта (ПОП); все последующие изменения вносились при строгом контроле над конфигурацией системы. САПО претерпела бесчисленное количество изменений после того, как была создана ее первая версия. Эти изменения тщательно рассматривались по мере развития проекта, но динамика изменений САПО привела к получению приемлемой

архитектуры с серьезным обоснованием правильности на ранних этапах жизненного цикла. САПО оказалась полезна при оценке изменений в общих программных интерфейсах, в ней также была отражена концептуальная архитектура Подсистемы общего назначения.

На рис. D.3 показана архитектура ПО с точки зрения ее стабильности. Из графиков видно, что архитектура подвергалась значительным изменениям на протяжении первых 20 месяцев выполнения проекта, после чего стабилизировалась. Большой пик в районе пятого месяца на графиках процессов и задач соответствует попытке обращения к более распределенному подходу. Как только эти эксперименты в области архитектуры завершились принятием соглашений, касающихся стратегий распределения, разработка САПО вернулась к исходному числу процессов. А вот разработка на уровне отдельных задач САПО стабилизировалась на новом более высоком числе задач. Основными проблемами, изучавшимися командой по разработке архитектуры, были достижение соглашений в области параллельного выполнения, накладные расходы на операционную систему, накладные расходы на создание динамических библиотек, разбиение на страницы, переключение контекстов и смешанный обмен сообщениями между процессами, задачами и узлами. Сложность этих взаимодействий в процессе выполнения привела к тому, что моделирование и эмуляция оказались неэффективными. Лишь благодаря демонстрациям на ранних этапах множества вариантов распределения позволили команде, занимающейся разработкой архитектуры, прийти к пониманию того, какие соглашения необходимы для выбора адекватного решения. Если изменения в структуру распределения вносятся на поздних стадиях проекта, последствия могут оказаться непредсказуемыми. Поскольку вносить изменения в межпрограммные интерфейсы и сообщения было легко, а сами интерфейсы и сообщения относились к прикладным интерфейсам более низкого уровня, количество таких изменений постоянно оставалось на невысоком уровне на протяжении всей контрольной точки критического обзора проекта (КОП).

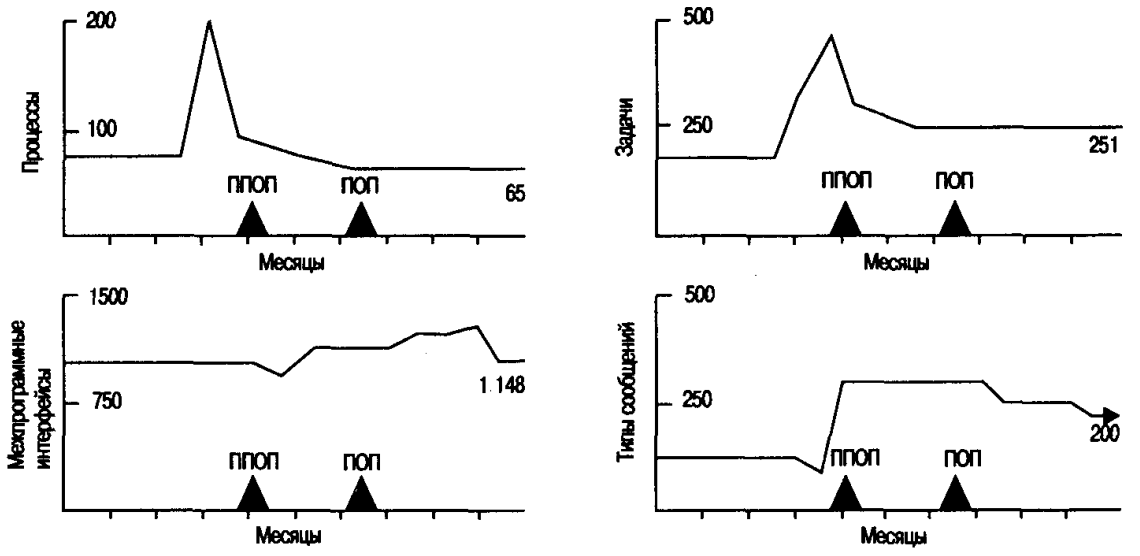


Рис. D.3. *Эволюция САПО Подсистемы общего назначения*

Свобода экспериментов с архитектурой доказала свою ценность с точки зрения возможности получения адекватной базовой архитектуры на ранних стадиях жизненного цикла. Это оказалось возможным прежде всего благодаря гибкости CSCI CAC.

Проект CCPDS-R был ориентирован на подход с упреждающей разработкой архитектуры. «Архитектурное» описание системы CCPDS-R давалось прежде всего с точки зрения процесса (см. главу 7). Это происходило из-за жестких требований к работе системы и риска, присущего некоторым разновидностям распределенной архитектуры первого поколения.

D.5 ОБЗОР ПРОЦЕССА

После заключения контракта разработка проекта CCPDS-R следовала стандартному для Министерства обороны жизненному циклу, в который входили рассмотрение требований к ПО, предварительный обзор проекта, критический обзор проекта и завершающее квалификационное тестирование. Жизненный цикл 12-месячной конкурентной разработки и жизненный цикл стадии полномасштабной разработки могут быть поставлены в соответствие стадиям итерационного процесса, описанным в главе 5. Рис. D.4 показывает это соответствие.

Для управления этой огромной работой по созданию ПО было определено шесть последовательных версий. На рис. D.4 представлены содержание версий и их перекрытие, а индивидуальные параметры версий и микропроцесс более подробно описываются в разделе D.5.1. Завершение каждой версии соответствует новой базовой версии Подсистемы общего назначения. С точки зрения макропроцесса на начальных контрольных точках основное внимание уделяется созданию базовой архитектуры. ПОП требует трех больших итераций по архитектуре, завершение которых совпадает с контрольными точками обзора требований к ПО (ОТПО), промежуточного ПОП (ППОП) и ПОП:

1. ОТПО-демонстрация: начальная реализуемость основных компонентов, а также базовые варианты использования для инициализации и взаимодействия между процессами.
2. ППОП-демонстрация: реализуемость архитектурной инфраструктуры для наиболее рискованных вариантов использования, включая следующие:
   - Пиковая нагрузка сценария оповещения о запусках ракет данными в случае внезапной массовой ракетной атаки Советского Союза
   - Сценарий управления пиковой нагрузкой при сбое системы и переходе с основного потока обработки на дублирующий поток без потери данных
3. ПОП-демонстрация: адекватное выполнение сценариев при пиковой нагрузке и остальных основных вариантов использования в рамках полномасштабной архитектурной инфраструктуры, включая другие компоненты критических потоков управления.

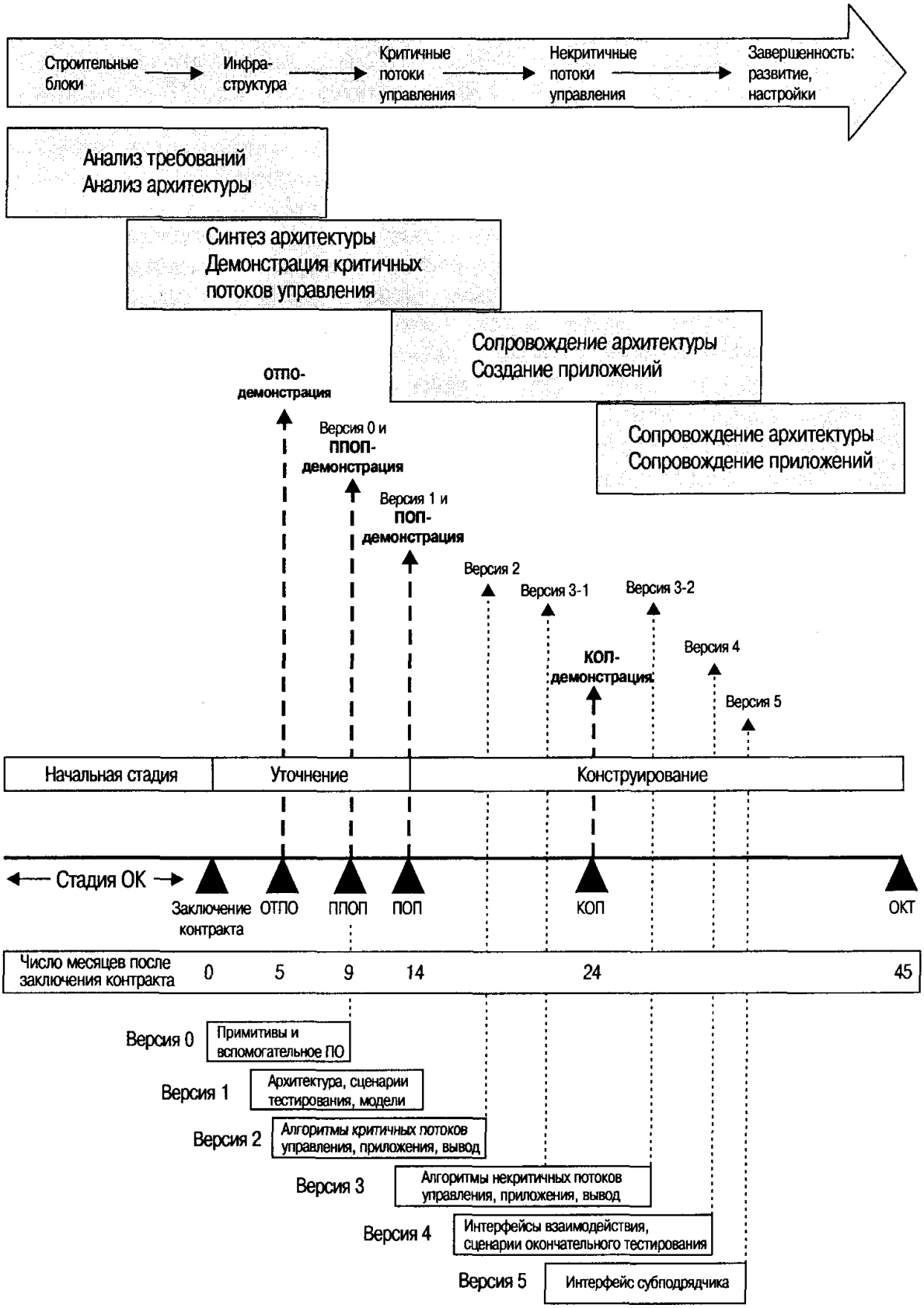


Рис. D.4. *Макропроцесс, контрольные точки и сроки проекта CCPDS-R*

КОП-демонстрация обновляет базовую архитектуру с тем, чтобы она соответствовала эквиваленту возможностей альфа-теста для полной архитектурной инфраструктуры и сценариев критических потоков управления. Эта система обеспечивала выполнение некоторого множества вариантов использования, которые позволяли пользователю частично выполнить свое задание.

В целом процесс создания ПО в рамках проекта CCPDS-R обладал четко определенным макропроцессом, аналогичным стадиям жизненного цикла (см. рис. 5.1). Каждая основная контрольная точка сопровождалась основательной демонстрацией возможностей и обычно включала в себя несколько текущих версий. Было бы более точным называть процесс разработки, использованный при реализации проекта, пошаговым, а не итерационным, хотя, как и в случае любой крупномасштабной системы, он с очевидностью имел черты и того и другого.

D.5.1 Управление рисками: содержание версии

Планирование содержания и сроков создания версий Подсистемы общего назначения привело к получению полезного и точного общего плана управления рисками. Необходимость в наличии подробного плана версии хорошо осознавалась командой управления, поэтому он был тщательно продуман еще на начальной стадии. Команда управления определяла ожидаемое реальное содержание версии по мере прохождения жизненного цикла, что давало более точные оценки сложности, риска, персонала и проектных решений. Такой эволюционирующий план был чрезвычайно важен, он позволял вносить некоторые коррективы в содержание версии и в сроки ее реализации по мере того, как первоначальные догадки становились объективными фактами.

На рис. D.5 подробно показаны сроки и содержание CSCI Подсистемы общего назначения. Рассмотрим детально содержание версий:

- Версия 0. Включает в себя основные компоненты, необходимые для построения скелетной архитектуры ПО. В нее входят разработка взаимодействия между задачами/процессами, выполнение общих задач и процессов, а также общие компоненты сообщений об ошибках. Версия 0 является также завершением исследований и разработки проекта, выполнявшихся параллельно стадии ОК (начальной стадии). CAC-компоненты представляют собой краеугольные камни архитектуры и спроектированы таким образом, чтобы их можно было повторно использовать во всех трех подсистемах CCPDS-R. Это очень сложные компоненты с высоким риском, к ним предъявляются жесткие требования по производительности, надежности и возможности повторного использования.
- Версия 1. Посвящена собственно «архитектуре». В нее входят полный набор задач (300 штук), процессов (70 штук), различных видов взаимодействия (1000 видов), состояний и переходов состояний,

CSCI	0	1	2	3	4	5	Итого
CAC	8	8	2	2			20
CC		33	25	102			160
КВ			23	27	20		70
ТИЭ		2	3	5			10
ООД			3	6	6		15
ВВД			5	31	37	7	80
Итого	**8**	**43**	**61**	**173**	**63**	**7**	**355**

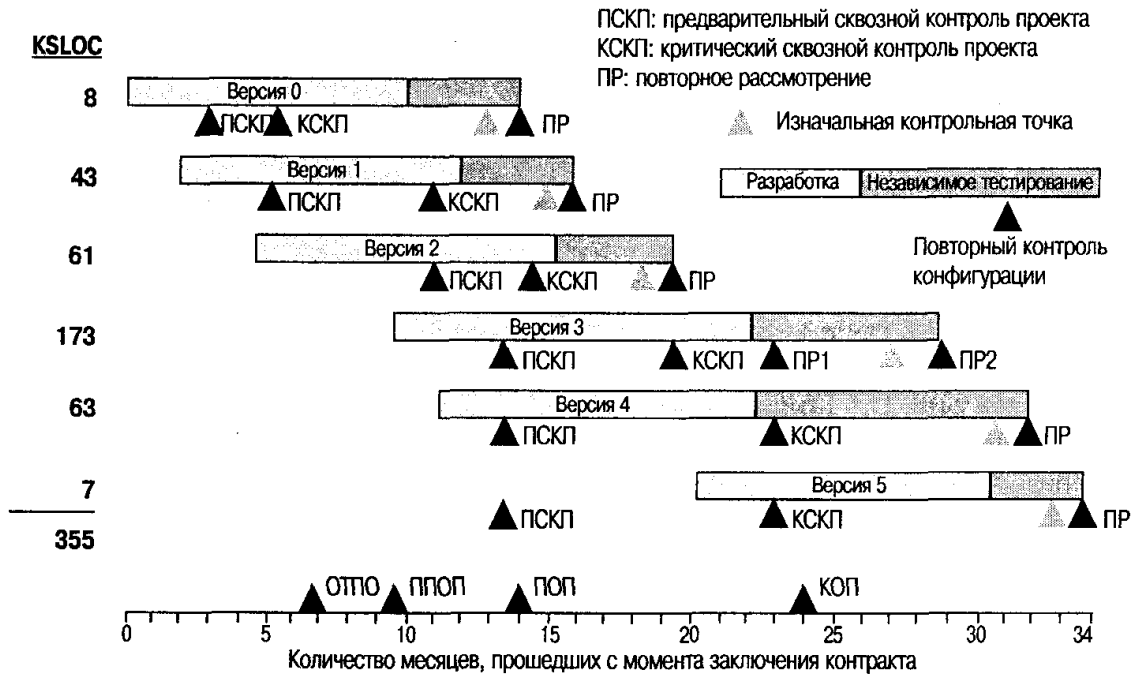


Рис. D.5. *Версии Подсистемы общего назначения*

необходимых для структурного решения архитектуры ПО CCPDS-R. Для того чтобы архитектура соответствовала полному циклу развития, в эту версию были добавлены все CAC-компоненты для инициализации, управления состоянием (реконфигурации) и инструментального оснащения. Для организации первоначальных демонстраций введены простой интерфейс пользователя и возможность включения в архитектуру сценариев тестирования. К моменту

завершения версии 1 можно продемонстрировать лишь некоторые критичные варианты использования; среди них инициализация архитектуры, ввод сценария тестирования для пропуска потока данных через систему и общие изменения конфигурации, например, переключение с основного потока на дублирующий.

- Версия 2. Это была первая версия, в рамках которой создавались компоненты, критичные для работы системы в целом, а также достигалась первоначальная возможность исполнения сценариев реальных заданий. Тремя основными рисками в сценариях выполнения задания являлись своевременность распределения базы данных для вывода, производительность (потребляемые ресурсы и точность) алгоритмов обработки данных, поступающих с радаров оповещения о запусках ракет, и реализация пользовательского интерфейса для некоторых сложных случаев вывода. По завершении версии 2 появлялась возможность исполнения некоторых практических вариантов, ориентированных на выполнение задания, включая поток обработки данных в самом плохом случае и управляющий поток в наихудшем случае (переключение с основного на дублирующий).
- Версия 3. Содержит наибольший объем кода, включая определение формата вывода, определения глобальных типов, спецификации представления, необходимые для обеспечения правильного исполнения операций, связанных с внешними интерфейсами. Большая часть кода создана автоматически «по рецептам» путем создания инструментов для генерации кода. К остальным компонентам, входящим в версию 3, относятся поддержка протокола интерфейса внешних взаимодействий, полный пользовательский интерфейс для операторов, пользовательский интерфейс для компьютеризованных рабочих мест, системные службы для реконфигурации заданий, восстановление баз данных, предварительная обработка данных в режиме оффлайн и алгоритмы обработки данных о ядерных взрывах. Изначально планировавшаяся в качестве одной большой версии, эта часть разработки позже разбивается на две более управляемые версии 3-1 и 3-2.
- Версия 4. Обеспечивает окончательную установку алгоритмов оповещения для спутниковых систем раннего обнаружения, окончательную установку средств вывода, позволяющих управлять заданием и получать информацию о его состоянии, а также окончательную установку процедуры обработки интерфейсов внешних взаимодействий.
- Версия 5. По ходу работы над Подсистемой общего назначения была добавлена версия 5 для обеспечения совместимости с некоторым внешним интерфейсом (созданным в результате выполнения другого контракта), поскольку сроки этой работы совершенно не укладывались в рамки изначально спланированной версии (версии 4). Это привело к тому, что создание внешнего интерфейса было выделено в абсолютно новую версию.

Последовательность версий, определенная для проекта CCPDS-R, является хорошим примером типичной последовательности версий, рекомендуемой в разделе 10.4.

D.5.2 Пошаговый процесс проектирования

Отдельные контрольные точки в рамках каждого этапа включали в себя предварительный сквозной контроль проекта (ПСКП), критический сквозной контроль проекта (КСКП) и повторное рассмотрение (ПР). Сроки достижения этих контрольных точек определялись в зависимости от сроков достижения контрольных точек более высокого уровня данного проекта (ОТПО, ППОП, ПОП и КОП) и объединялись с ними. На рис. D.6 показан общий вид жизненного цикла отдельной версии и то, на что направлялись основные усилия.

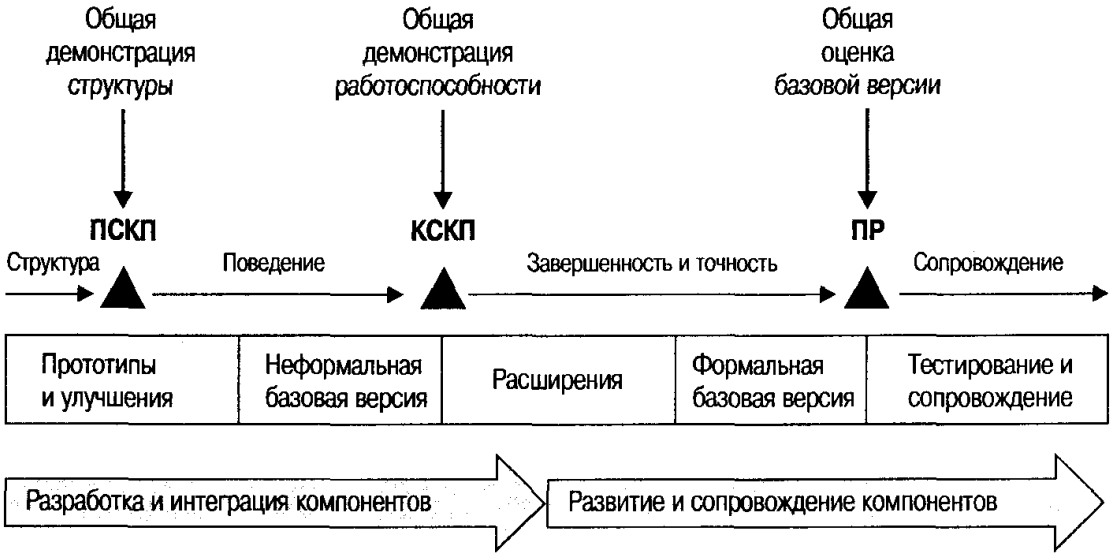


Рис. D.6. *Основные виды деятельности в рамках отдельной версии*

В рамках каждой версии существует четко определенная последовательность процедур сквозного контроля проекта. Такой контроль являлся неформальным детальным техническим инспектированием промежуточных продуктов разработки. В нем принимали участие все, кто был в этом заинтересован, включая других разработчиков, ответственных за тестирование и даже лиц из сторонних заинтересованных организаций (заказчиков, пользователей и персонала, ответственного за проектирование систем). Это участие обычно ограничивалось небольшим числом компетентных людей, обычно 10 – 20 человек. Самое серьезное внимание при таком контроле уделялось наиболее важным компонентам, интерфейсам архитектуры и основным вопросам, иными словами, тщательного рассмотрения заслуживало приблизительно 20% от всего ПО. В охвате всех требований и всех компонентов необходимости не было.

Сквозной контроль проекта был неформальным и сопровождался интенсивным обменом мнениями, для него была характерна открытая критика. При этом удавалось обходиться без всяких скучных мероприятий. Технические проблемы принимались как руководство к действию и отслеживались до полного разрешения. Процедуры ПСКП и КСКП длились один-два дня, в каждой из них принимали участие менеджеры по различным CSCI, ответственные за представление соответствующих материалов.

Предварительный сквозной контроль проекта

Первоначальные создание прототипов и разработка завершались ПСКП и демонстрацией основных возможностей. Главное внимание при таком контроле уделялось атрибутам структуры тех компонентов, которые разрабатывались на данном этапе. Повестка дня в основном была для каждой версии своя, однако в общем случае в нее включались следующие темы для каждого CSCI:

- Обзор: обзор CSCI, интерфейсы, компоненты и метрики.
- Компоненты: сквозной контроль каждого основного компонента, демонстрирующий интерфейс его исходного кода, распределение требований к СТПО (спецификации требований к ПО), текущие метрики, рабочая концепция для ключевых сценариев использования, план проведения независимого тестирования, а также условия возникновения и реакции на ошибки.
- Демонстрация: основное внимание сосредоточивалось на проверке управляющих интерфейсов всех компонентов в рамках интегрированной архитектуры.

Критический сквозной контроль проекта

Работа в рамках проектирования конкретной версии заканчивалась процедурой КСКП и демонстрацией возможностей, которая позволяла увидеть ключевые параметры работоспособности компонентов, созданных в данной версии. В то время как ПСКП сосредоточивался на описательной стороне разработки, при КСКП главное внимание уделялось законченности компонентов и тому, как их поведение при использовании будет соотноситься с заранее определенными требованиями к работоспособности. Повестка дня в основном была для каждой версии своя, однако в общем случае в нее включались следующие темы для каждого CSCI:

- Обзор CSCI: интерфейсы, компоненты и параметры; сводка изменений, внесенных со времени проведения ПСКП; решение всех проблем, возникших при ПСКП; сценарии интеграционного тестирования версии.
- Компоненты: сквозной контроль каждого основного компонента, демонстрирующий интерфейс их исходного кода, распределение требований к СТПО, текущие метрики, рабочая концепция для

ключевых сценариев использования, план проведения независимого тестирования, а также условия возникновения и реакции на ошибки.

- Демонстрация: основное внимание сосредоточивалось на проверке выполнения критических потоков управления.

Сквозной контроль кода

Тщательный сквозной контроль кода использовался также для распространения знаний и опыта на весь проект и для того, чтобы способствовать созданию самодокументируемого исходного кода. Некоторые авторы писали исходный код, который оказывался настолько удобочитаемым, что удостаивался оценки «самодокументируемый». Менеджеры по CSCI и главный разработчик ПО занимались координацией необходимого контроля кода и его распределением между различными авторами, при этом преследовались такие цели:

- Более широкое распространение «самодокументируемого» стиля написания программ
- Обнаружение ошибок в программах, которые не всегда могут быть выловлены компиляторами и инструментами для анализа кода
  - Наименования объектов, стиль программирования и стиль комментариев: являются ли они удобочитаемыми?
  - Неоправданно усложненные объекты или методы: не существует ли более простых подходов?
  - Повторное использование: не создавалось ли ПО на заказ там, где имеются повторно используемые компоненты?
  - Потенциальные проблемы с производительностью: существуют ли потенциально неэффективные реализации?
- Уменьшение объема исходного кода, подлежащего рассмотрению при сквозном контроле проекта на более высоком уровне
- Знакомство неопытных сотрудников с продуктами, созданными экспертами, и наоборот

Типичная проверка кода выполнялась одним проверяющим, который был ограничен двумя часами подробного анализа с использованием инструментария для просмотра исходного кода в онлайновом режиме. Результаты проверки оформлялись в виде одностраничного описания относящихся к делу комментариев, которое передавалось автору программы, CSCI-менеджеру и главному разработчику ПО. Главный разработчик ПО отвечал за выявление глобальных тенденций, определение необходимых улучшений в инструментарии, предназначенном для анализа программ, и за подготовку информации об извлеченных уроках к соответствующим проверкам или иным техническим мероприятиям по обмену опытом.

Повторные рассмотрения

Повторные рассмотрения на самом деле не являются рассмотрениями; обычно это ведущаяся в течение месяца работа, в процессе которой завершается независимое тестирование компонентов, после чего происходит возврат к контролю конфигурации, интеграционному тестированию версии и функциональному тестированию.

Контрольные точки, использовавшиеся при пошаговом проектировании CCPDS-R, могут служить хорошим примером второстепенных контрольных точек (см. раздел 9.2). Сочетание сквозного контроля и инспекций сосредоточивалось на 20% компонентов, которые потенциально могли дать наибольшую отдачу от вложенного труда. Вообще говоря, истинная ценность сквозного контроля проекта и инспекций заключалась во взаимных контактах между различными подгруппами разработчиков и в методической координации процессов. В ходе этих встреч удалось вскрыть небольшое количество серьезных качественных пороков (в отличие от вскрытых при демонстрациях), однако эти контрольные точки позволили довести до автоматизма процесс обмена технической информацией.

D.5.3 Эволюция компонентов

В проекте CCPDS-R в качестве унифицированного формата для ведения проекта на протяжении всего жизненного цикла использовался язык программирования Ada. Такая унификация позволяла получать значения метрик, определяющих прогресс в разработке ПО, непосредственно из исходных файлов. Применение Ada в качестве языка разработки базировалось на специальном пакете разработки, содержащем объекты, чьи названия имели префикс TBD (to be defined — подлежащие определению в дальнейшем). Пакет TBD-объектов включал в себя TBD-типы, TBD-константы, TBD-значения и TBD-процедуру, отображающую строки исходного кода и связанные с ними комментарии, которые совместно друг с другом могли быть использованы как «заполнители» (placeholders), зарезервированные для пока еще ненаписанных сегментов программы. Процедура объявлялась следующим образом:

```
TBD_Statements (Number_Of_Statements: In Integer);
```

Такое объявление требовало, чтобы параметр, определяющий число операторов и приблизительно вычисленный для данного сегмента программы, был описан в соответствующих комментариях. Строки исходного кода с обращениями к TBD-объектам рассматривались как строки ADL (Ada design language — язык разработки Ada); строки, не имеющие ссылок на TBD, рассматривались как исходные строки на языке Ada. В таблице D.2 дается пример типичной эволюции компонента.

Таблица D.2.
Типичная эволюция компонента от создания до повторного рассмотрения

Представление	Модуль программы	Тип	Ada	ADL	Всего	%
В момент создания	Весь пакет		6	122	128	5
	Inm_Erm_Procedures	Пакет	2	122	124	2
С точки зрения ПСКП	Весь пакет		47	101	148	32
	Inm_Erm_Procedures	Пакет	24	19	43	56
	All_Node_Connections	Процедура	3	19	22	14
	Create_Inm_Erm_Circuits	Процедура	4	8	12	33
	On_Node_Connections	Процедура	3	7	10	30
	Perform_Reconfiguration	Процедура	6	2	8	75
	Perform_Shutdown	Процедура	4	3	7	57
	Process_Error_Messages	Процедура	3	43	46	7
С точки зрения КСКП	Весь пакет		87	48	135	65
	Inm_Erm_Procedures	Пакет	30	11	41	73
	All_Node_Connections	Процедура	16	0	16	100
	Create_Inm_Erm_Circuits	Процедура	8	4	12	67
	On_Node_Connections	Процедура	9	0	9	100
	Perform_Reconfiguration	Процедура	6	2	8	75
	Perform_Shutdown	Процедура	6	1	7	86
	Process_Error_Messages	Процедура	12	30	42	29
С точки зрения повторного рассмотрения	Весь пакет		137	0	137	100
	Inm_Erm_Procedures	Пакет	42	0	42	100
	All_Node_Connections	Процедура	16	0	16	100
	Create_Inm_Erm_Circuits	Процедура	12	0	12	100
	On_Node_Connections	Процедура	9	0	9	100
	Perform_Reconfiguration	Процедура	8	0	8	100
	Perform_Shutdown	Процедура	7	0	7	100
	Process_Error_Messages	Процедура	43	0	43	100

Эволюция основных компонентов выглядит следующим образом:

- На момент создания только интерфейс (в части его спецификации) определяется посредством исходных строк на языке Ada и соответствующих комментариев. Оценочное количество SLOC для всего компонента обычно задается в виде значения строки `TBD_Statements`.
- К моменту проведения ПСКП внутренняя структура компонента постепенно заполняется описаниями данного компонента и оценками размеров программных модулей более низких уровней с

использованием множественных обращений к TBD_Statements. В этот момент времени 30% SLOC компонента обычно бывают написаны на языке Ada и 70% — на ADL.

- К моменту КСКП большинство интерфейсов программных модулей и описаний полностью написано на языке Ada, хотя некоторая детальная обработка по-прежнему использует TBD_Statements в качестве заполнителей. Вообще говоря, компоненты на уровне КСКП обычно состоят на 70% из строк на языке Ada и на 30% из строк на ADL. Существует правило, гласящее, что к моменту КСКП не должно быть обращений к TBD_Statements со значениями, большими 25.
- К моменту повторного рассмотрения строка TBD не должна появляться где бы то ни было в исходных файлах. Это соответствует завершению реализации.

Иногда эти правила нарушались, однако эволюция большинства компонентов следовала данному образцу весьма точно. Кроме того, использовались подробные стандарты стиля, которые формировали основу для сквозного контроля кода на ранних этапах, и требования к автоматическому аудитору кода, который выполнял проверки на соответствие бесчисленным стандартам до начала повторного рассмотрения.

Одним из побочных следствий применения Ada в качестве языка разработки проекта CCPDS-R являлось то, что изменяющиеся исходные файлы всегда имели унифицированный формат представления, который позволял с легкостью получать информацию об объеме выполненной (исходные строки на языке Ada) и незаконченной (TBD_Statements) на данный момент работы. Хотя исходные строки, написанные на языке Ada, необязательно были завершены — постольку поскольку в ходе дальнейшей разработки в них могли вноситься изменения — они давали относительно точную оценку объема выполненной работы. Полный комплект файлов, касающихся разработки, по всем командам разработчиков мог быть подвергнут анализу в любой момент с целью получения ясности относительно общего прогресса работ. Был создан измерительный инструмент, позволяющий сканировать исходные файлы на языке Ada и собирать статистику о количестве завершенных строк на языке Ada и о количестве TBD_Statements. В качестве выходных получались данные, аналогичные представленным в таблице D.3.

Этот измерительный инструмент и стандарты написания программ для проекта CCPDS-R обеспечили сбор значений метрик по CSCI и по этапам так, что прогресс мог отслеживаться с нескольких точек зрения. Метрики прогресса разработки, описанные в разделе D.7.1, определялись ежемесячно на основании выходных данных, получаемых в результате работы этого инструмента, и представлялись при сквозном контроле проекта каждым разработчиком компонентов для определения суммарных метрик и для обсуждения иерархии компонентов.

Таблица D.3.
Итоговая статистика по параметрам ТИЭ CSCI на 10-ом месяце

Элементы	Всего	Разработано	Закодировано
Компоненты самого высокого уровня	40	39	33
Компоненты более низкого уровня	13	13	10
Суммарное число программных модулей	494	484	459
	Строк исходного кода: 18 494	ADL: 1858 10% TBD	Ada: 16 636 90% завершено

Данный инструмент позволял руководству проекта производить некоторые важные измерения прогресса непосредственно на основе исходного кода базовой версии ПО. Разработчики ПО лишь следовали стандартам разработки ПО при создании своих исходных файлов и при поддержании их в допустимых форматах. Один раз в месяц весь исходный код обрабатывался с помощью инструментов и рассматривался с различных точек зрения для определения прогресса. Получаемые параметры оказывались полезными не только менеджерам, но и разработчикам, например, для обоснования того, почему требуется больше ресурсов или почему необходимо изменить приоритеты определенных видов деятельности. Как показано в главе 13, принятие метрик как менеджером, так и разработчиком-практиком, а также непосредственное их извлечение из рабочих продуктов оказываются критичными для успеха подхода, использующего метрики.

D.5.4 Процесс пошагового тестирования

Общие требования к тестированию были чрезвычайно сложными, однако структура проекта CCPDS-R позволяла использовать вполне управляемую и ясную программу проведения тестов. Значительная часть неформального тестирования была естественным побочным продуктом ранних демонстраций архитектуры и того требования, что все компоненты должны содержаться в формате, пригодном для компиляции.

Поскольку на протяжении всего жизненного цикла в качестве главного формата применялась компилируемая форма языка Ada, большинство проблем, связанных с интеграцией, таких как непротиворечивость типов данных, устаревание программных модулей и взаимозависимость программных модулей, выявлялось и разрешалось на этапе компиляции.

Однако неформального тестирования, свойственного процессу демонстраций, оказывалось совершенно недостаточно для того, чтобы убедиться в полном соответствии требованиям и в достижении ожидаемой надежности для системы такой степени важности. Поэтому была задумана строгая последовательность тестирования, состоявшая из пяти различных видов деятельности: независимое тестирование, интеграционное тестирование версии, тестирование надежности, функциональное тестирование и окончательное квалификационное тестирование.

1. Независимое тестирование (НТ). На команды разработчиков возлагалась ответственность за независимое тестирование компонентов до того, как они формально включались в базовую версию ПО, находящуюся под управлением конфигурацией и подвергавшуюся дальнейшему тестированию. В процессе НТ обычно тестировался отдельный компонент (который мог состоять из нескольких компонентов более низкого уровня) в независимой среде. Этот уровень тестирования соответствует тестированию завершенности и соблюдению граничных условий в степени, максимально возможной для НТ.

2. Интеграционное тестирование версии (ИТВ). Это небольшой тест, позволяющий убедиться в том, что ранее продемонстрированные возможности по-прежнему реализуются должным образом. Последовательность ИТВ-тестов служит прежде всего способом качественной оценки того, можно ли завершать повторную проверку. Повторная проверка данной версии может длиться днями и неделями в зависимости от ее размера или от процента вновь созданных компонентов. Задачей ИТВ является не проверка на соответствие требованиям, а создание стабильной, надежной базовой версии. Оно неформально, динамично и сосредоточено на выявлении ошибок и противоречий. В процессе ИТВ проверяется следующее:
   - Ранее продемонстрированные потоки управления могут быть успешно повторены.
   - Ранее определенные недостатки устранены.
   - Интерфейсы межкомпонентного взаимодействия протестированы полностью.
   - Базовая версия является достаточно стабильной для того, чтобы тестирование на соответствие требованиям оказалось эффективным.

3. Тестирование надежности. В результате выполнения ИТВ и повторной проверки получалась стабильная базовая версия, которая подвергалась широкому тестированию в неурочное время с максимальной нагрузкой в течение длительных промежутков времени по случайным, но реалистичным сценариям. Этот вид тестирования был разработан для выявления рутинных непостоянных ошибок основного хода разработки. Тестирование надежности проводится в течение максимально возможного времени и обычно тогда, когда ресурсы не заняты ничем другим (по ночам и выходным).

4. Функциональное тестирование (ФТ). Основное внимание уделяется проверке на соответствие определенным подмножествам требований по многим CSCI посредством демонстраций и тестирования реализаций различных вариантов использования (называемых функциональными потоками управления).

5. Окончательное квалификационное тестирование (ОКТ). Эти тесты эквивалентны ФТ за исключением того, что они производят проверку соответствия тем требованиям, для которых это может быть

сделано, только если имеется вся система. Например, требование наличия 50%-ного резерва пропускной способности не может быть проверено, пока система не будет функционировать как единое целое.

Общий план построения подсистемы определялся распределением всех критичных по надежности компонентов (компонентов, которые могут приводить к ошибкам типа 0) по версиям 0, 1 и 2. На рис. D.7 показаны общий ход работ по тестированию и базовые версии, подвергавшиеся тестированию в соответствии с планом. Такая последовательность построения базовых версий давала максимум времени для того, чтобы созданные на ранних этапах компоненты критичных потоков управления успевали приобрести законченный вид. Эти компоненты подвергались также более полному тестированию, что повышало уверенность в их готовности к фактическому использованию. На тестирование выделялось время, необходимое для получения эмпирического значения среднего времени наработки ПО на отказ (MTBF), которое демонстрировалось и принималось заказчиком. Например, первые версии Подсистемы общего назначения содержали все необходимые компоненты для управления состоянием потока обработки, для выявления сбоев, восстановления после сбоев, интерфейсов взаимодействия с операционной системой и для распределения данных в режиме реального времени. Одним словом, в них было включено 90% компонентов из тех, что способны привести систему к критичным сбоям, не позволяющим выполнить задание.

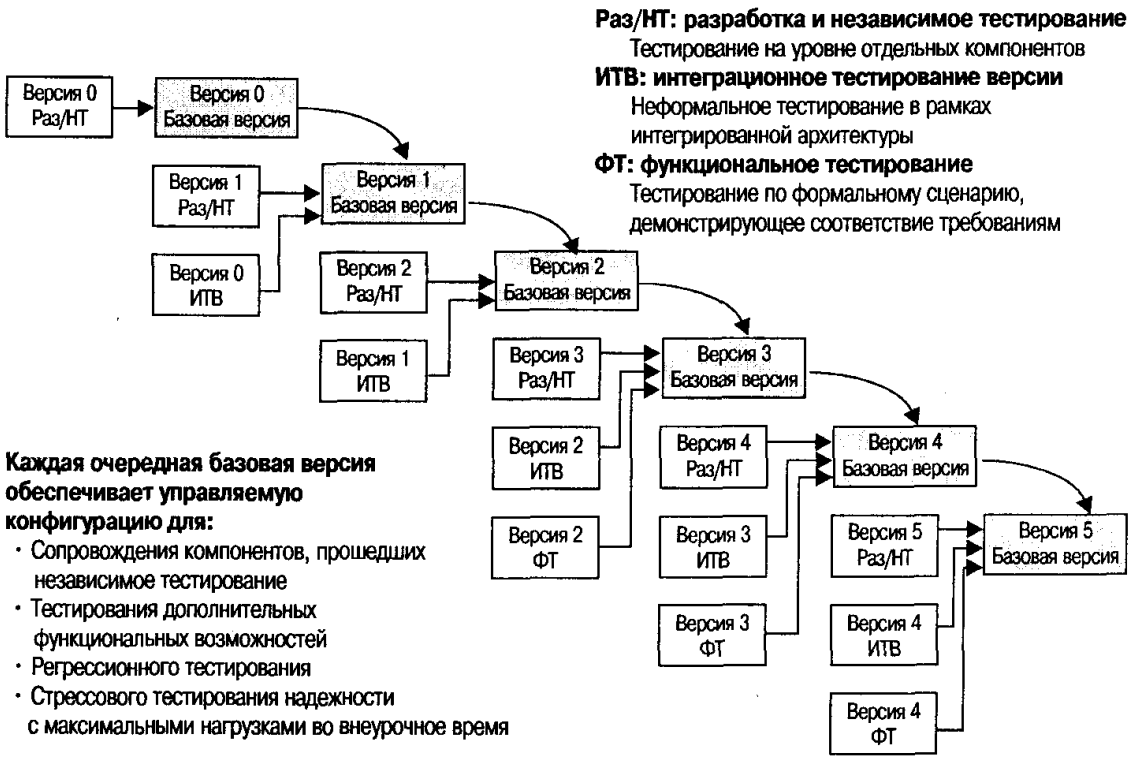


Рис. D.7. *Последовательная эволюция базовой версии и направление, в котором ведется тестирование*

Последовательность версий CCPDS-R и программа тестирования являются хорошими примерами первоочередного разрешения наиболее важных рисков. На ранних этапах жизненного цикла достигалась также стабильность архитектуры, при этом становилось возможным серьезное тестирование надежности. Эта стратегия позволила применить полезные метрики завершенности, вроде тех, что были определены в разделе 13.3, для демонстрации пользователю реального MTBF.

D.5.5 Рабочие продукты, регламентированные стандартом DOD-STD-2167A Министерства обороны

Разработка ПО CCPDS-R должна была соответствовать стандарту DOD-STD-2167A, который в настоящее время уже устарел. Не вдаваясь в подробности относительно необходимой документации, рассмотрим основополагающий подход к созданию документов, использовавшийся в этом проекте. Описания проектных данных в 2167A определяют формат и содержание документов. Допускалось внесение существенных изменений, которые бы отвечали применяемому подходу и соответствовали использованию языка Ada в качестве языка разработки и языка реализации. Основными были следующие изменения:

1. Применение исходных файлов на языке Ada в качестве единственного однородного формата проекта на протяжении всего жизненного цикла и самодокументируемое изменение этих файлов. Тексты на языке Ada удобочитаемы, и исключалась лишняя работа для подготовки отдельных подробных описаний проекта, которые с неизбежностью отдалялись от реализации.

2. Организация последовательности тестирования и выходной документации, касающихся содержания данной версии, определялась подмножеством вариантов использования (которые называют последовательностями рабочих операций и сценариями), а не CSCI. Тестирование, основанное на этих сценариях, охватывало компоненты из нескольких CSCI. Оно организовывалось для каждой версии посредством плана тестирования ПО, процедуры тестирования ПО и последовательности отчетных документов о тестировании. Последовательности документов определялись для каждого ИТВ (по одной для каждой версии), для каждого ФТ (для версий 2, 3 и 4) и для ОКТ (одна завершающая всеобъемлющая последовательность тестов). В каждую последовательность тестов вовлекались компоненты из нескольких (незавершенных) CSCI, поскольку процесс интеграции шел постоянно.

3. Создание документации по тестированию независимых модулей в виде самодокументируемого однотипного ПО. Это обеспечение рассматривалось так же, как и остальной рабочий исходный код с тем, чтобы оно имело тот же формат и можно было регулярно обновлять

его для автоматизированного выполнения регрессионного тестирования. Такая же концепция использовалась при проведении тестирования по ИТВ- и ФТ-сценариям: вместо того чтобы разрабатывать документацию по процедуре тестирования, в процессе создания CCPDS-R генерировались самодокументируемые сценарии тестирования, которые являлись полноценным программным продуктом. Поскольку внесение в них изменений управлялось точно так же, как и внесение изменений в остальное ПО, они всегда содержались в обновленном виде для проведения автоматизированного регрессионного тестирования.

В таблице D.4 представлена вся документация, которая возникла в результате адаптации стандарта 2167A и соответствующих рабочих продуктов (см. главу 6). Подход, содержащийся в стандарте 2167A, был абсолютно неэффективен, даже с учетом адаптации (хотя он оказался намного эффективнее, чем тот подход, который использовался в большинстве традиционных проектов). С самого начала было ясно, что бремя создания документации непомерно, однако отход от традиционного процесса был признан чересчур рискованным. В таблицу D.4 включена только документация ПО; в нее не вошли документы, касающиеся инженерных аспектов систем (безопасность, человеческий фактор, надежность) и сообщества пользователей (план освоения, материально-техническое обеспечение, обучение). Эти документы также требуют затрат от организации-разработчика на создание и сопровождение, даже с учетом того, что в рамках проекта CCPDS-R основная ответственность за них была возложена на других.

Таблица D.4.
Программные рабочие продукты проекта CCPDS-R

Номер	Документ, создание которого оговорено в контракте	Соответствующие рабочие продукты (см. главу 6)
1	Системные спецификации	Общая концепция
6	Спецификации требований к ПО (СТПО)	Окончательные спецификации версии, по 1 на каждый CSCI
1	Описание проекта системы (ОПС)	Описание архитектуры
6	Общее описание проекта (ООП)	UML-модели разработки, по 1 на каждый CSCI
42	Файл разработки ПО (ФРПО)	Комплект рабочих продуктов реализации, по 1 на каждый компонент
6	Спецификации программного продукта (СПП)	Окончательный комплект рабочих продуктов реализации Комплект рабочих продуктов внедрения, по 1 на каждый CSCI
4	План демонстраций (согласно стандарту 2167A не требуется)	Спецификации версий по основным контрольным точкам, по 1 на каждую основную демонстрацию

Таблица D.4. (продолжение)
Программные рабочие продукты проекта CCPDS-R

Номер	Документ, создание которого оговорено в контракте	Соответствующие рабочие продукты (см. главу 6)
4	Отчет о демонстрациях (согласно стандарту 2167A не требуется)	Описание версий по основным контрольным точкам, по 1 на каждую основную демонстрацию
9	Файл с данными по тестированию (ФДТ)	Описания версий, по 1 на каждую последовательность тестов ИТВ, ФТ и ОКТ для каждой версии
4	План тестирования ПО (ПТПО)	Спецификации версий Комплект рабочих продуктов проектирования, тестовые модели
4	Процедура тестирования ПО (ПРТПО)	Комплект рабочих продуктов реализации
4	Отчет о тестировании ПО (ОТПО)	Описания версий
1	План разработки ПО (ПРПО)	План разработки ПО
1	Руководство по программным стандартам и процедурам (РПСП)	План разработки ПО
48	Обзор по управлению проектом (ОУП)	Оценки состояния
3	Руководство пользователя ПО (РППО)	Руководство пользователя, по 1 для каждой роли

Одним из ключевых рабочих продуктов среди представленных в таблице D.4 является файл разработки ПО (ФРПО). В проекте CCPDS-R это был не документ, а директория специальной структуры с информацией, доступной в онлайновом режиме, большая часть которой хранилась в виде допускающего компиляцию самодокументируемого исходного кода, написанного на языке Ada. Содержимое ФРПО состояло из нескольких разделов, которые подвергались изменениям, описанным в таблице D.5.

Таблица D.5.
Эволюция файла разработки ПО

Раздел	Состояние в момент ПСКП	Состояние в момент КСКП	Состояние в момент ПР
Требования	В общих чертах	Распределенное	Отслеживаемое
Рассмотрение компонентов	Завершено	Завершено	Завершено
Программный модуль самого верхнего уровня	Ada	Ada	Базовая версия на языке Ada
Модули более низкого уровня	Ada/ADL	Ada/ADL	Базовая версия на языке Ada
НТ-план	Черновик	Завершено	Завершено
Программа НТ-тестирования	Частичная демонстрация	Черновик	Базовая версия на языке Ada

Таблица D.5. (продолжение)
Эволюция файла разработки ПО

Раздел	Состояние в момент ПСКП	Состояние в момент КСКП	Состояние в момент ПР
Результаты НТ-тестирования	Частичная демонстрация	Частичная демонстрация	Базовая версия на языке Ada
Регистрация SCO	Отсутствует	Отсутствует	Начальная
Метрики	Начальные метрики	Обновленные метрики	Обновленные метрики
Результаты работы аудитора кода	Отсутствуют	Начальные	Окончательные
Замечания/отказы	Отсутствуют	Отсутствуют	По мере необходимости

Подход к рабочим продуктам в проекте CCPDS-R был преобразован и стал напоминать подход, представленный в главе 6. Изначально большинство рабочих продуктов было на бумаге. По мере того как заказчик начал проявлять больший интерес к демонстрируемым рабочим продуктам и базовым версиям компонентов продукта, потребность в бумажных документах снизилась — недостаточно, но хоть в какой-то степени. Большим достижением явился переход на полностью электронный ФРПО, для которого стандарты по разработке и кодированию поддерживали самодокументирование рабочих продуктов. Отдельные рабочие продукты для документирования проекта и кодирования больше не требовались. Одной из застарелых проблем CCPDS-R была необходимость графического описания проекта на высоком уровне. Она обеспечивалась в документации системного проекта и в проектных документах ПО самого высокого уровня за счет применения специализированного текста и графики. Подобные представления были неоднозначными, зачастую неактуальными и сложными для понимания. Использование нотации языка Unified Modeling Language, подхода к архитектуре, описанного в главе 7, средств визуального моделирования и поддержки «круговой» разработки позволило бы существенно усовершенствовать подход к представлению проекта и избежать большого количества бессмысленной работы.

D.6 ОЦЕНКА, ОСНОВАННАЯ НА ДЕМОНСТРАЦИИ

Традиционные обзоры разработок определяют стандартные темы таким образом, что это приводит к чрезмерно обширным обзорам, только небольшая часть которых оказывается по-настоящему важной или доступной для понимания всем участникам. Например, рассмотрение всех требований с равной степенью детализации является неэффективным и непродуктивным. Требования не равны; одни являются критичными для

эволюции архитектуры в процессе разработки, в то время как другие критичны лишь для некоторых компонентов. Процедура рассмотрения ПО в рамках проекта CCPDS-R позволила повысить эффективность проведения разработок, обзоров и достижения согласия между заинтересованными сторонами двумя способами: распределяя техническую широту и глубину обзора по менее масштабным процедурам сквозного контроля внутри разработки и сосредоточивая внимание при проведении обзоров по достижении основных контрольных точек на наиболее важных соглашениях, касающихся разработки. Более того, уделять основное внимание при таких обзорах выполняемым демонстрациям оказалось более понятным и конкретным способом рассмотрения для разнородной группы представителей заинтересованных сторон.

Во многих традиционных проектах создавались демонстрационные версии или контрольные задачи для решения отдельных проблем в рамках проекта (например, имитации интерфейса пользователя или критичного алгоритма). Однако «базовая версия проекта» обычно представлялась на бумаге в виде обзоров проекта или проектных документов. И хотя заинтересованным сторонам было легко принять эти рабочие продукты как правильные, они были неоднозначными и не могли быть исправлены непосредственно с помощью внесения изменений. С учетом распространенного отношения к обзорам проекта, смысл которого заключался в том, что проект «невиновен, пока не доказано обратное», форматы представления позволяли создавать заслуживающий доверия фасад и утверждать, что проект «невиновен». Напротив, процесс обзора проекта ПО в CCPDS-R основывался на демонстрациях, для которых были необходимы ощутимые доказательства того, что прогресс в разработке и в архитектуре ведет к получению приемлемого качества. Демонстрации при рассмотрениях проекта позволяли получать такие доказательства, так как они предъявляли работающую в критичных сценариях текущую версию архитектуры.

Множество качеств базовой архитектуры становилось видимым при каждом конкретном рассмотрении проекта. Демонстрации обеспечивали как минимум ясное понимание целостности архитектуры и составляющих ее компонентов, рисков, которые могут возникать в процессе работы, а также понимание рабочей концепции системы и ключевых вариантов использования.

В проекте CCPDS-R уроки, извлеченные из неформального сквозного контроля проекта (и соответствующих неформальных демонстраций), отслеживались посредством предпринимаемых действий. Обзоры проекта по достижении основных контрольных точек включали в себя и брифинги, и демонстрации. На брифингах кратко излагались общие сведения о проекте в целом, а также важные результаты, полученные при сквозном контроле, и давался общий обзор задач, сценариев и ожидаемых результатов демонстрации. В процессе рассмотрения проекта демонстрация являлась кульминацией реального процесса рассмотрения, проводившейся командой разработчиков. В последовательность действий по проведению демонстраций входили разработка плана, определение набора критериев оценки, интеграция компонентов в нечто реально

выполняющееся и создание тестовых драйверов, сценариев и используемых только для тестирования компонентов. Хотя планы проведения демонстрации не были тщательно проработанными (они занимали обычно от 15 до 35 страниц), в них содержались цель демонстрации, реальные критерии оценки результатов, сценарии выполнения и общее описание программных и аппаратных средств, подлежащих демонстрации.

Существует интересная отличительная особенность оценки работоспособности при использовании подхода, основанного на демонстрациях. Если при традиционном подходе практически всегда оценки сначала оптимистичны, а потом становятся хуже, то при современном подходе, основанном на демонстрациях, оценки зачастую бывают сначала пессимистичны, но потом улучшаются.

При проведении демонстраций в рамках проекта CCPDS-R были извлечены следующие уроки:

- **Раннее создание сценариев тестирования способствует высокому ROI.** Инвестирование в создание некоторых критичных сценариев тестирования на ранних этапах служит двум целям. Во-первых, это способствует тому, что некоторое важное подмножество требований «оформляется» во вполне осязаемом виде. Сценарии тестирования являются побудительным мотивом определенных контактов и переговоров с пользователями, которые способствуют улучшению понимания требований на ранних этапах жизненного цикла. Во-вторых, выполнение этих работ вовлекает команду, ответственную за тестирование, в создание среды для демонстраций и тестирования на ранних стадиях, что обеспечивает чрезвычайную зрелость среды к тому моменту, когда проект достигает этапа полномасштабного тестирования.

- **Планирование и проведение демонстраций выявляет важные риски.** Переговоры, касающиеся содержания каждой демонстрации и связанных с ней критериев оценки, служат для того, чтобы сфокусировать внимание команды по разработке архитектуры, команды управления и внешних заинтересованных сторон на критичных требованиях на ранних этапах и при выполнении работ по созданию архитектуры. Вместо того чтобы иметь дело с полным детальным проектированием и трассировкой всех 2000 требований, команда сосредоточивается на 20-ти или около того ведущих требованиях к проекту.

- **Инфраструктура, инструментальное оснащение и вспомогательное ПО для демонстраций имеют высокий ROI.** В начале проекта казалось, что эти демонстрации потребуют значительных вложений в компоненты, которые могут быть использованы только для демонстраций. В большинстве случаев очень незначительная часть этой работы оказывалась пригодной только для демонстраций. Большая часть работы приводила к созданию компонентов, которые затем повторно использовались в последующем при независимом тестировании, интеграционном тестировании версии или

функциональном тестировании. Для того чтобы дать представление об объеме дефектных компонентов, скажем, что для ППОП-демонстраций было написано приблизительно 72 000 SLOC. Из них всего лишь около 2000 SLOC (заглушки и отладочные сообщения) было впоследствии выброшено за ненадобностью.

- **Работы по подготовке демонстраций приводят к достижению соглашений по проекту.** Интеграция для последующей демонстрации обеспечивает установление своевременной обратной связи по вопросам, касающимся важных атрибутов проекта и его уровня зрелости. Работы по подготовке демонстраций обычно требовали участия от 10 до 12 разработчиков для интеграции компонентов в единую архитектуру. Они сталкивались с множеством препятствий, переделок, и им пришлось несколько раз вносить изменения как в компоненты, так и в архитектуру. Эта работа осуществлялась на протяжении месяца по большей части поздно вечером. Что на самом деле происходило в процессе всенощных интеграций-отладок-переделок-изменений, так это подробное, чрезвычайно эффективное рассмотрение проекта. Координируя работы, мне удавалось получать из первых рук информацию о том, каковы сильные и слабые стороны архитектуры, какие компоненты уже созрели, а какие пока еще сырые, как следует расставить приоритеты в работе, которую предстоит выполнить после окончания демонстрации.

- **Проблемы в работоспособности, обнаруженные на ранних этапах, приводят к ранним улучшениям архитектуры.** Первые две демонстрации охватывали большое число функциональных возможностей и в процессе выполнения имели более низкую работоспособность, чем требовалось. Критерии оценки демонстраций не сильно отличались от окончательных требований к работоспособности. Оглядываясь назад, можно сказать, что это было контрпродуктивно, поскольку приводило к тому, что определенная часть наблюдавших за ходом выполнения контракта ожидала, что критерии оценки демонстрации и требования будут схожи. Заказчик и руководство компании TRW были изначально обеспокоены такой ситуацией, однако простые решения и существенный прогресс, достигнутый в последовательно проводившихся демонстрациях, привели к улучшению понимания.

Реализация демонстраций в качестве основного промежуточного продукта разработки хорошо понятна. В разделе 9.1 уделено мало внимания обсуждению вопросов координации множества заинтересованных сторон. Однако при наличии нескольких заинтересованных сторон проведение оценок, основанных на демонстрациях, может усложниться. В последующих разделах подробно обсуждается опыт, полученный при разработке проекта CCPDS-R.

ППОП-демонстрация

Промежуточная ПОП-демонстрация Подсистемы общего назначения по достижении основной контрольной точки имела три важные цели:

1. Реальная оценка целостности разрабатываемой архитектуры ПО посредством создания САПО-прототипа
2. Реальная оценка понимания критичных требований посредством создания наихудшего возможного сценария оповещения о ракетных запусках
3. Выявление архитектурных рисков при пиковых нагрузках в сценарии оповещения о ракетных запусках (работоспособность при обработке данных в наихудшем случае, соответствующем массовой ракетной атаке Советского Союза) и в сценарии обнаружения сбоя и последующего восстановления (наихудший случай при управлении обработкой, связанный со сбоем в основном потоке обработки и с переключением в реальном времени на дублирующий процесс обработки, находящийся в «горячем» резерве)

Для культуры создания ПО в рамках проекта CCPDS-R эти цели являются очевидными. Демонстрации не были показухой, предназначенной для того, чтобы произвести впечатление на заказчика отличными результатами и минимальным количеством нерешенных проблем. (Для этого не были предназначены ни сквозной контроль, ни рассмотрения состояния проекта руководством, ни основные контрольные точки.) Демонстрации всегда были честной инженерной деятельностью с честолюбивыми целями, с открытым обсуждением компромиссов и с открытым подходом к обоснованию оценок, касающихся прогресса и качества. По результатам демонстраций обычно производились изменения в требованиях, планах и разработке в одинаковой степени; все эти три аспекта подвергались изменениям на протяжении жизненного цикла.

Работы по подготовке демонстраций обычно охватывали шестимесячный период, при этом основное внимание в течение первых трех месяцев уделялось планированию. Весьма немногие представители коллективов заинтересованных сторон принимали участие в определении формальных критериев оценки. На рис. D.8 показано общее расписание ППОП-демонстрации; подробно представлен также период интенсивной интеграции, длившийся два месяца перед началом демонстрации.

Первые три месяца планирования, которые включали в себя создание чернового плана, рассмотрение этого плана правительством и его комментарии, а также создание окончательного плана, могли бы быть завершены за одну неделю правильно подобранной командой с участием всех заинтересованных сторон. Последовательность рассмотрений, которая происходила на самом деле, определялась требованиями контракта. Поскольку и компания TRW, и заказчик впервые столкнулись с подходом, основанным на демонстрациях, обе стороны не представляли, как наилучшим образом организовать процесс, и согласились на консервативный подход. Эта демонстрация явилась первой попыткой создания

Рис. D.8. *Работы и сроки их выполнения для первой демонстрации в рамках проекта CCPDS-R*

полномасштабной САПО. Соответственно, это была первая (и самая неудачная) попытка выполнения общей интеграции Подсистемы общего назначения. Последующие демонстрации обычно характеризовались менее продолжительными при равной интенсивности работами по интеграции, длящимися 4 – 5 недель.

Область действия ППОП-демонстрации

Основная область действия ППОП-демонстрации определялась положением о выполнении работ по проекту CCPDS-R:

> *Подрядчик обязан продемонстрировать следующие возможности NORAD. Демонстрация 1: вспомогательные функции системы, инициализация системы, поведение при сбое и восстановление системы, реконфигурация системы, ввод тестовых сообщений и регистрация данных.*

Эти возможности были понятны и заказчику, и компании TRW. Они представляли ключевые компоненты и варианты использования, необходимые для достижения целей.

1. Вспомогательные функции системы – это компоненты ПО САС общего назначения, предназначенные для повторного использования во всех трех подсистемах. Эти компоненты составляли основу архитектуры. Они включали в себя службы взаимодействий между процессами, общий контроль за приложениями (диспетчеры задач и

процессов), CAC-утилиты (списки, службы имен и последовательностей) и общие службы мониторинга и сообщений об ошибках. Этих компонентов оказалось достаточно для того, чтобы продемонстрировать любой выполняемый поток управления.

2. Регистрация данных (CC CSCI). Эта возможность необходима для обработки некоторых результатов и связана с производительностью.
3. Компоненты, отвечающие за ввод тестовых сообщений (ТИЭ CSCI), позволяли вводить сообщения в любой объект системы таким образом, что тому придавались свойства общего тестового драйвера.
4. Инициализация системы являлась основным вариантом использования (на рис. D.8 это Стадия 1), который мог продемонстрировать существование согласованной скелетной архитектуры ПО и безошибочную работу значительной части служб системы. Один из рисков, связанных с производительностью, заключался в требовании инициализировать ПО большого объема с распределенной архитектурой, содержащее как сделанные на заказ, так и коммерческие компоненты, в течение заданного времени.
5. Второй сценарий (Стадия 2) предназначался для формирования пиковой нагрузки по передаче сообщений, которая бы вызывала внутреннюю передачу сообщений по всей системе некоторым реальным образом. Для выполнения этого сценария требовалось, чтобы для всех программных объектов были «смоделированы» правильные, но простые заглушки для обработки сообщений. Эти простые программы на языке Ada завершали выполнение потока обменом фиктивными сообщениями, читая и отправляя сообщения таким образом, каким это должно было осуществляться при пиковых нагрузках. ПО для обработки сообщений прототипов предназначалось для приема входящих сообщений и передачи их далее по компонентам, составляющим САПО. Сюда включался весь сколько-нибудь значимый обмен сообщениями, начиная от получения сообщений от внешних датчиков и заканчивая обновлением выходных сообщений о ракетных запусках, как для основного, так и для дублирующего потоков. Кроме того, сюда же был включен обмен всеми вспомогательными сообщениями, связанными с мониторингом состояния, сообщениями об ошибках, мониторингом производительности и регистрацией данных.
6. Одним из наиболее рискованных сценариев являлся сценарий сбоя и восстановления системы (Стадия 3), поскольку для его осуществления требовалось выполнение весьма изощренного набора интерфейсов, контролирующих управление состояниями и переход из одного состояния в другое, в логической сети, состоящей из сотен программных объектов. Основной задачей такого варианта использования было создание искусственного сбоя в рабочем объекте основного потока управления с целью вызова всей последующей цепочки

событий: обнаружение сбоя, уведомление о сбое, полностью оформленный переход от основного потока к дублирующему, закрытие основного потока. Все эти изменения состояний в сети должны были происходить без прекращения обслуживания операторов оповещения о ракетных запусках. Реконфигурация в данном конкретном случае означала восстановление режима после сбоя. При сбое системы дублирующий поток должен инициализироваться таким образом, чтобы влияние разовых сбоев было минимальным. В готовой системе восстановление происходило немедленно вслед за сбоем.

Критерии оценки ППОП-демонстрации

Основные критерии оценки ППОП формировались на основании требований, оценки рисков и соглашений относительно разработки:

- Для всех стадий:
  - Не должно происходить критичных ошибок.
- Стадия 1:
  - Система должна инициализироваться не более чем за 10 мин.
  - Инициализация системы должна выполняться с единственного терминала.
  - После инициализации число процессов, задач и межпрограммных интерфейсов обязано в точности совпадать с тем, которое должно быть в текущей базовой версии САПО.
- Стадия 2:
  - Средняя загрузка процессора каждого узла в наихудший момент сценария с 20-минутной пиковой нагрузкой не должна превышать 30%.
  - Не должно быть ошибок, связанных с дублированием или потерей сообщений.
  - Все выводимые данные должны быть получены не позднее, чем через одну секунду после их ввода.
  - Процесс приема сообщений должен поддерживать такую интенсивность ввода, которая соответствует предполагаемому для данного сценария уровню.
  - Регистрация данных не должна приводить к неожиданным изменениям состояния или к сообщениям об ошибках, при этом должны быть зарегистрированы все введенные сообщения.
- Стадия 3:
  - Оператор должен иметь возможность имитации сбоя в любом объекте.
  - Сообщение об ошибке должно быть получено в течение 2 секунд с момента сбоя.

- Переключение с основного потока на дублирующий должно завершаться не более чем через 2 секунды после сбоя без потери данных.
- Закрытие основного потока после сбоя и его повторная инициализация в качестве дублирующего потока должны завершаться не позднее чем через 5 минут после сбоя.
- Зарегистрированные данные должны соответствовать ожидаемым изменениям состояния без фатальных ошибок, кроме специально сымитированного сбоя.

Кроме того, существовало еще 23 критерия оценки для рассмотрения менее важных частных возможностей и промежуточных результатов. Здесь они не приводятся.

Результаты ППОП-демонстрации

Результаты ППОП-демонстрации оказались весьма плодотворными. Соответствие 31-му критерию оценки из 37-ми было признано удовлетворительным. Шесть критериев не были реализованы, среди них — три важных критерия из тех, что обсуждались выше. Это было расценено как очень серьезная проблема, которая требовала немедленного проведения новой разработки и повторной демонстрации. Наиболее серьезной проблемой оказалась чрезмерная загрузка процессора при выполнении сценария пиковой нагрузки. Хотя пороговое значение было определено как 30%, реальная загрузка составляла 54%. Это свидетельствовало о существенных непроизводительных издержках архитектуры, операционной системы и сетевого ПО. Поскольку этот риск всегда осознавался при разработке системного ПО для проекта CCPDS-R, данной проблеме было уделено самое серьезное внимание. Для анализа производительности были созданы пять различных задач, а также была поставлена цель продемонстрировать улучшение производительности в процессе следующего обзора управления проектом после того, как эти пять задач будут решены.

В упрощенном виде эти пять задач формулируются следующим образом:

1. Изменение сценария. Текущий сценарий тестирования фактически задавал пиковую нагрузку на 33% хуже, чем реальная пиковая нагрузка. Процессы обмена внутренними сообщениями оказались хуже, чем самый плохой из реальных вариантов (например, каждое сообщение приводило к «сигналу тревоги», результатом чего был чрезмерный и ненужный обмен информацией). ППОП-демонстрация заставила компанию TRW, заказчика и пользователя уточнить описание сценария для самого трудного задания в конкретных и объективных терминах. Это привело также к тому, что команда разработчиков архитектуры стала лучше понимать порядок движения сообщений и соглашения, касающиеся оптимизации. Отдача от выполнения этих работ никогда не определялась количественно, но она наверняка была значительной.

2. Настройка параметров буферизации взаимодействий между процессами (ВМП). CAC-компоненты имеют множество опций для оптимизации работоспособности. Несмотря на то, что за тот месяц, который завершал работы по интеграции, было сделано множество локальных оптимизаций, существовала определенная необходимость в более глобальном анализе с целью извлечения уроков из различных примеров обмена сообщениями.

3. Расширение сетевых транзакций. Обмен сообщениями между двумя узлами, очевидно, являлся узким местом, поскольку в текущей версии операционной системы (DEC VMS 4.7) никак не использовалась возможность симметричной многопроцессорной обработки процессорами VAX. Переход на VMS 5.0 обеспечивал существенное повышение общей производительности за счет данного компонента.

4. Улучшение работоспособности ВМП-компонента. Очевидное узкое место в компоненте, отвечающем за CAC-взаимодействие между процессами, оказывало существенное влияние на одну из возможностей оптимизации. Группа, отвечающая за демонстрацию, расценила это как порок разработки, нуждающийся в переделке. (Предварительное решение уже находилось в разработке.)

5. Повышение надежности ВМП-компонента. ППОП-демонстрация позволила выявить еще один серьезный недостаток: внезапное резкое повышение числа сообщений могло приводить к возникновению ошибок в работе. Чрезвычайно жесткий сценарий сделал этот порок очевидным. Для системы с такими жесткими ограничениями по надежности, какие накладывались на CCPDS-R, этот недостаток обязательно следовало зафиксировать, хотя при эксплуатации системы он мог бы никогда не проявиться. Фиксация такого рода проблем в тот момент могла пройти безболезненно, однако незамеченный до поздних этапов выполнения проекта этот недостаток мог привести к значительному браку и огромному объему дефектов и доработок.

Описанные пять задач в точности соответствовали критичным проблемам, остававшимся нерешенными на момент демонстрации. Это вызвало сильную тревогу как у руководства компании TRW, так и у заказчика; обе стороны надеялись на то, что демонстрация не выявит нерешенных проблем. Несмотря на это, обеим сторонам понравились процесс демонстрации и то беспрецедентно глубокое знакомство с реальным прогрессом разработки и с соглашениями по разработке, понимание требований и оценка рисков, которые они смогли получить. Общий уровень беспокойства заинтересованных сторон снизился после выполнения поставленных задач и повторной демонстрации, которая была проведена приблизительно через месяц после ППОП-демонстрации. Хотя изначальная цель – загрузка процессора не более чем на 30% – не была достигнута, команда показала гибкость архитектуры и наличие возможностей для оптимизации: ей удалось снизить общую загруженность процессора с 54% до 35%. Благодаря такой позитивной тенденции все успокоились по поводу того, что требования к работоспособности будут непременно

выполнены за счет непосредственной оптимизации разработки и перехода на более новые версии операционной системы.

Результаты ППОП-демонстрации можно разделить на видимые и формальные. В качестве руководителя, ответственного за процесс, архитектуру и данную демонстрацию, я видел множество незаметных результатов. За период ночных работ по интеграции и тестированию, который длился 8 недель и в течение которого были расставлены многие приоритеты, разрешены многие вопросы, произведен мозговой штурм целого ряда проблем, заинтересованные стороны постоянно получали отчеты о состоянии работ, команды разработчиков обрели мотивацию для решения амбициозных задач, удалось извлечь множество полезных уроков:

1. За этот период времени удалось эффективно проанализировать разработку. Демонстрация явилась результатом рассмотрения, выполненного командой разработчиков и представленного заинтересованным сторонам в качестве осязаемого свидетельства прогресса. После ее завершения осталось всего пять нерешенных проблем. За этот восьминедельный период было обнаружено, решено и закрыто не менее 50 проблем, касающихся разработки. Такое раннее устранение дефектов, содержащихся в требованиях, процессе, инструментарии и в самой разработке, имело документально не подтвержденную, но существенную отдачу в виде отсутствия огромного количества дефектов на более поздних этапах, которые бы обязательно всплыли, не реши мы их в период ранней демонстрации.

2. Благодаря ежедневному участию в этой работе, мне удалось понять, в чем заключаются слабые и сильные стороны разработки и почему. Например, когда мы сталкивались с какими-либо проблемами в компонентах, ответственному разработчику хватало нескольких часов, чтобы найти соответствующее решение. Другие компоненты не поддавались так легко, и поиск решения зачастую требовал нескольких дней. К моменту завершения работ по проведению демонстрации мне было известно, где внесение изменений оказалось простым делом (что обычно свидетельствовало о хорошей разработке компонентов), а где представляло определенные сложности (по очень многим причинам). Такая информация помогала при определении структуры рисков для перспективного планирования, при распределении сотрудников и установке приоритетов для тестирования.

3. Демонстрация в большой степени способствовала созданию работоспособной команды, поскольку имела вполне конкретную цель, а разработчики могли трудиться в удобном для них режиме: материал для работы всегда имелся в наличии.

4. Детальное понимание технических аспектов и объективное обсуждение соглашений по разработке оказались бесценными для установления доверительных отношений со всеми заинтересованными сторонами, включая заказчика, пользователя и руководство компании TRW. Мы были вооружены фактами и цифрами, а не субъективными спекуляциями.

Реакция правительства на ППОП-демонстрацию

Формальная ППОП-демонстрация представляла собой значительную смену парадигмы по сравнению с традиционной процедурой рассмотрения проекта. Соответственно, между компанией TRW и военно-воздушными силами существовали большая напряженность и беспокойство по поводу подробных критериев оценки для этой демонстрации. Следующие абзацы, выделенные курсивом, заимствованы из окончательного плана, утвержденного компанией TRW. Это хорошая подборка вопросов, которые вероятнее всего будут выделяться среди остальных при использовании организацией такого процесса в первый раз. Она позволяет также проникнуться духом демонстраций.

После тщательного изучения комментариев к правительственному предварительному плану Демонстрации 1 были сформулированы следующие замечания, которые подводят итог утверждению плана Демонстрации 1 и изменений, внесенных по сравнению с предыдущей версией:

1. *Из утверждаемого плана исключены все упоминания о требованиях для того, чтобы избежать намеков на намерение соответствовать, подтверждать или демонстрировать любые требования. Проверка на соответствие этим требованиям выполняется строгим и регламентированным способом специальной организацией, ответственной за тестирование. Работы по проведению демонстрации должны быть направлены на интенсификацию разработки, выполняться с минимальным количеством документации с целью раннего понимания осуществимости разработки и ее прогресса. Компания TRW намеревается максимизировать пользу от демонстрации как от одного из видов деятельности по разработке и хотела бы избежать ее превращения в менее полезную деятельность по интенсивной разработке документации.*

2. *В некоторых правительственных комментариях говорилось о необходимости дальнейшей детализации требований, разработки и т.д. Эта информация не нужна для плана демонстрации. Она либо повторяет сведения, содержащиеся в других документах (СТПО, ОПС, пакетах по проверке разработки), либо была предоставлена двумя неделями ранее демонстрации при проведении процедуры неформального тестирования. Включение большего количества информации в один документ (и это справедливо для любого документа), возможно, и облегчит работу проверяющему, однако это будет излишним, потребует дополнительных затрат времени и усложнит создание документа, уменьшая тем самым техническое содержимое рассматриваемого продукта, находящегося в разработке.*

3. *В свете подхода правительства к связи между требованиями и демонстрацией, критерии оценки, представленные в данном плане, должны быть тщательно изучены. Мы считаем, что критерии оценки являются точными и глубокими в смысле определения осуществимости разработки, особенно на столь раннем этапе жизненного цикла. Хотя мы*

остаемся открытыми для внесения в эти критерии оценки конструктивных изменений, мы считаем, что их изменение с целью более точного соответствия Спецификациям системы является в данном случае неподходящим. Точка зрения требований и точка зрения данной демонстрации различны и плохо соотносятся между собой.

4. *Исходный код демонстрируемых компонентов не поставлялся вместе с планом, как того требует положение о работе. Общий объем продемонстрированных компонентов составляет половину от общего объема, и это соотношение постоянно и быстро меняется. Вместо того чтобы поставлять весь исходный код, тем, кто заинтересован в просмотре того или иного кода, была предоставлена возможность запросить конкретные компоненты на рассмотрение. Весь исходный код доступен для просмотра в организации-разработчике в процессе демонстрации.*

Как уже упоминалось ранее, общая реакция правительства на ППОП-демонстрацию была весьма позитивной, несмотря на выявление пяти серьезных проблем. Когда по истечении одного месяца компания TRW продемонстрировала решение всех этих задач, реакция правительства стала более чем позитивной. Объективные представления, открытая дискуссия относительно соглашений и понятность проблем в разработке, требованиях и функционировании системы привели к установлению исключительных отношений между заинтересованными сторонами. Представители заказчика и пользователей потребовали проведения дополнительной демонстрации для своего высшего руководства, а среди заинтересованных сторон витал дух успеха, разделяемый всеми. Это событие доказало свою важность: с этого момента все стремились поддерживать репутацию данного проекта как флагмана, с которого следует брать пример того, как правильно создавать ПО.

D.7 ГЛАВНЫЕ МЕТРИКИ

Подход к определению метрик изначально разрабатывался исключительно для целей управления проектом и для достижения соответствия требованиям контракта. После того как поставленные цели были достигнуты, они стали практическим примером для изучения. Выполнение проекта CCPDS-R никогда даже не приближалось к оптимальному; в процессе выполнения постоянно совершалось большое количество ошибок. Подобное утверждение справедливо и для программы по определению метрик: иногда измерялось не то, что надо, иногда измерялось не так, как надо. Она не способствовала ранней интерпретации, и в ней применялись ручные методы там, где была необходима автоматизация. Тем не менее работа с метриками привела к улучшению командного труда, к улучшению процессов, к лучшему пониманию рисков и, безусловно, к созданию более эффективного продукта. На ранних стадиях проекта существовало сопротивление со стороны управления, со стороны практических разработчиков и даже со стороны наблюдателей за ходом выполнения

контракта. По истечении первого года, после нескольких усовершенствований в интерпретации, автоматизации, представлении и определении ощущалась практически единодушная поддержка. Все стороны использовали объективные данные из программы по определению метрик для обоснования своих планов, рисков, направлений разработок и результатов.

Все представленные метрики Подсистемы общего назначения извлекались непосредственно из ежемесячных обзоров по управлению проектом. Ни одно из этих значений не создавалось постфактум. Хотя программа по определению метрик и являлась требованием, содержащимся в контракте, правительство не определило, какие именно метрики должны использоваться. Это было оставлено на усмотрение подрядчика с тем, чтобы команда, выполняющая проект, самостоятельно приняла на себя ответственность за выбранную программу по определению метрик.

Компания TRW сформулировала четыре задачи программы по определению метрик:

1. Обеспечение данными для оценки текущих тенденций выполнения проекта и определения того, на что следует обратить внимание при управлении проектом
2. Обеспечение данными для планирования последующих этапов и создания других подсистем
3. Обеспечение данными для оценки относительной сложности достижения соответствия окончательным требованиям по качеству
4. Обеспечение данными, позволяющими определить, какие требуются усовершенствования процесса, и обосновать их необходимость

Ниже приводятся конкретные примеры метрик, рекомендованных в главе 13. Дается несколько примеров метрик для определения прогресса, а также качественных показателей дефектов, доработок и завершенности. Описываются основы, необходимые для автоматизации; они требуют некоторых интересных технических подходов, которые заключены непосредственно внутри рабочих продуктов, связанных с проектированием и кодированием.

D.7.1 Прогресс разработки

Точное измерение прогресса разработки при наличии нескольких параллельных видов деятельности, находящихся на различных стадиях, являлось сложной проблемой для команды, управляющей созданием Подсистемы общего назначения. Значительные усилия пришлось затратить на формирование согласованного подхода, который предоставил бы точную информацию относительно состояния на уровне подсистем и состояния версий. Целью было получение взвешенной оценки, в которую включалось бы следующее:

- Метрики Ada/ADL. Позволяли довольно точно определять непосредственные показатели технического прогресса. Сами по себе эти метрики абсолютно точно отражали прогресс в разработке и реализации. Однако они плохо подходили для описания завершенных частей контракта и финансового состояния.
- Метрики добавленной стоимости. Позволяли довольно точно определять финансовое состояние и готовые к поставке заказчику части контракта. Вообще говоря, они являются плохими показателями реального технического прогресса.

Как и в случае большинства других метрик ПО, оба эти подхода изначально давали неточные оценки абсолютного прогресса. Однако они являлись превосходными оценками относительного прогресса, если отслеживались регулярно (в нашем случае – ежемесячно). По мере накопления опыта работы с этими метриками абсолютные оценки постепенно настраивались для предсказания успеха или риска. Общие оценки сводились в единую диаграмму. На рис. D.9 показан итоговый прогресс на самом верхнем уровне для каждой отдельной версии и для Подсистемы общего назначения в целом. Длина заштрихованного участка внутри каждой версии относительно пунктирной линии (относящейся к текущему месяцу) определяет, опережает ли выполнение существующее расписание или отстает от него. Например, на рисунке изображено состояние после 17 месяца: НТ-тестирование версии 2 отстает от графика на один месяц, разработка версии 3 опережает график на один месяц, разработка

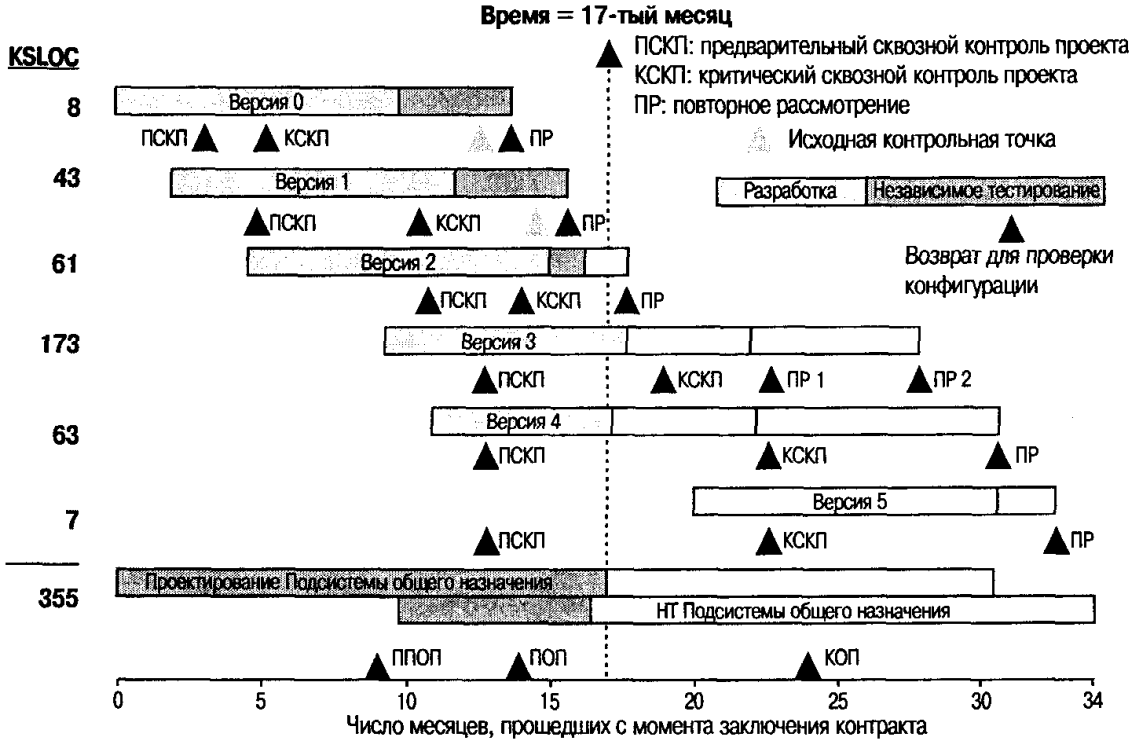


Рис. D.9. *Общий прогресс разработки*

Подсистемы общего назначения соответствует расписанию, а НТ-тестирование Подсистемы общего назначения отстает от графика на один месяц. Заштрихованные области – это оценка главного разработчика, который объединял ежемесячные значения метрик прогресса с ежемесячными значениями метрик финансового состояния в некую обобщенную (а потому, в некотором смысле, субъективную) оценку.

Ежемесячный сбор значений метрик обеспечивал необходимое для управления детальное понимание прогресса, увеличения объема кода и других показателей, достигнутых на каждой из версий. Метрики собираются по каждой версии и по CSCI с тем, чтобы иметь возможность рассмотрения под различными углами зрения. Менеджеры каждого отдельного CSCI собирали и оценивали свои метрики, прежде чем они сводились воедино для проекта в целом. Этот процесс являлся объективным, эффективным и осмысленным. Самый нижний уровень оценок `TBD_Statements` был, конечно, субъективным, однако они определялись наиболее осведомленными людьми: непосредственными разработчиками. Оценки хранились в формате исходного кода. В этом случае возрастала вероятность того, что в данном виде рабочих продуктов будет храниться самая последняя информация. Такой процесс позволял также непротиворечиво и единообразно сравнивать прогресс по различным направлениям проекта.

На рис. D.10 представлены ежемесячные оценки прогресса для Подсистемы общего назначения и для каждой версии. Планируемый объем изменений основывался на грубом средневзвешенном подсчете для каждой версии, выполнявшемся согласно указаниям, данным в разделе D.5.3: 30% объема создается к моменту ПСКП и 70% объема – к моменту КСКП. В целом Подсистема общего назначения практически полностью соответствовала своему плану за одним исключением. Прогресс, достигнутый к моменту ППОП (намного опережая график), отразил неожиданное позитивное влияние инструментария, генерирующего исходный код. С его помощью для САПО было сгенерировано более 50 000 SLOC.

Соответствие работы планам менялось в зависимости от конкретной версии. Прогресс, достигнутый для Подсистемы и для каждой версии, оценивался ежемесячно внутренним руководством и заказчиком в процессе обзоров управления проектом. Метрики прогресса являлись объективным механизмом и согласованным языком для описания изменений, вносимых в планы и архитектуру, проблем, возникающих при разработке, рисков при составлении графиков и других связанных с управлением аспектов. Объективность такого подхода являлась основной составляющей неантагонистических отношений, установившихся между всеми заинтересованными сторонами.

Все понимали, что, хотя значения метрик были не точны на ранних стадиях жизненного цикла, они были верны. Абсолютные значения редко когда оказывались важными. Более важными являлись относительные тенденции, и по мере развития процесса точность всех метрик возрастала. К моменту ПОП значения метрик стали краеугольным камнем при обмене информацией в рамках проекта.

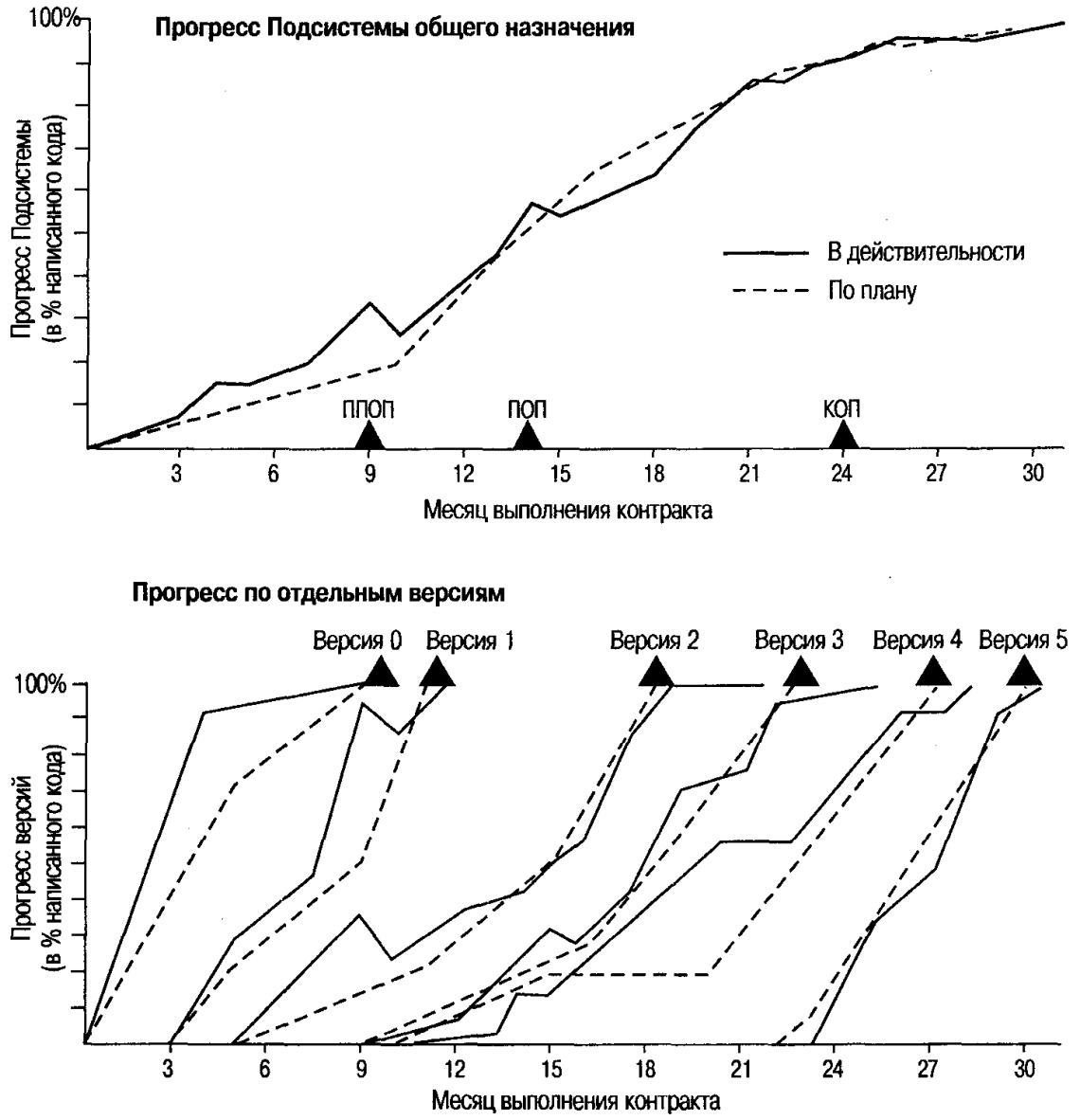


Рис. D.10. *Прогресс в разработке Подсистемы общего назначения*

D.7.2 Прогресс в тестировании

Организация, осуществлявшая тестирование, должна была создать тесты интеграции версий и тесты на соответствие требованиям (некоторые НТ-, ФТ- и ОКТ-тесты). Тестирование интеграции версий оказалось менее эффективным с точки зрения выявления проблем, чем ожидалось. ИТВ-тесты должны были содержать полный набор процедур тестирования интеграции — от базовых возможностей до особых граничных условий. Большая часть этой работы, в частности основные потоки, перекрывалась с работами по интеграции для демонстрации. Соответственно, ИТВ-тесты зачастую дублировали подготовку к демонстрациям, что было менее эффективно по стоимости, чем если бы деятельность по подготовке к демонстрациям была совмещена с ИТВ, а ответственность за нее

возложена на организацию, выполняющую тестирование. В таблице D.6 представлены результаты ИТВ этапа 2, которые отражают интегрированное состояние продукта. Однако на планирование, подготовку и проведение ИТВ было затрачено больше усилий, чем требовалось. Совмещение подготовки к демонстрациям с деятельностью по проведению ИТВ позволило бы меньшему числу сотрудников сделать работу лучше. Такой подход позволил бы увеличить степень интеграции (являясь составной частью работ по подготовке демонстраций) перед проведением повторной проверки и более эффективно выполнить обратное тестирование после проведения повторной проверки с целью убедиться в том, что все предыдущие проблемы разрешены.

Таблица D.6.
Характеристики SCO для ИТВ-тестирования версии 2

Источник проблем	Малый (< 1 часа)	Умеренный (< 1 дня)	Большой (> 1 дня)	Всего
Интерпретация требований	5			5
Проблемы при независимом тестировании	3	4	2	9
Проблемы с интерфейсами	9	2	1	12
Неправильное выполнение	1			1
Желательное расширение (это не проблема)	3			3
Несовместимая конфигурация	3	2		5
В сумме	24	8	3	35

Таблица D.7 и рис. D.11 позволяют взглянуть на метрики прогресса с различных точек зрения, которые применялись при планировании и отслеживании программы тестирования в проекте CCPDS-R. На рисунке изображен график зависимости прогресса относительно планируемого для тестирования соответствия требованиям. НТ-, ФТ- и ОКТ-тесты являлись источниками вариантов тестирования, использовавшимися организацией-разработчиком ПО. За НТ отвечали команды разработчиков, но оно должно было проводиться в формальной среде управления конфигурацией и под контролем (визуальным наблюдением) персонала, ответственного за тестирование. ФТ состояло из функционально связанных между собой групп сценариев, которые демонстрировали соответствие требованиям, охватывающим сразу несколько компонентов. ОКТ-тесты позволяли определять такие аспекты соответствия требованиям, которые не могли быть показаны до полного создания системы. Количественные требования к производительности (КТП) охватывали все CSCI.

Формальное НТ-тестирование (проверка на соответствие требованиям, выполняемая в форме независимого тестирования) оказалось более трудным, чем планировалось. Это произошло прежде всего из-за того, что спецификации требований и обзоры проекта оказались перегружены деталями, касающимися разработки, а также процедурами согласования.

Проведение формального НТ-тестирования тщательно контролировалось правительством и заняло чрезвычайно много времени на подготовку обзора. Правительство потребовало использовать подробнейшие процедуры тестирования для множества отдельных деталей проекта, которые на самом деле не следовало рассматривать как требования. В горячке разработки выполнение НТ-процедур редко когда оказывалось доступным за 30 – 60 дней до повторной проверки, как это предусматривал контракт для любых видов тестирования на соответствие требованиям. Процесс формального НТ-тестирования явился одной из основных причин того, что повторные рассмотрения постоянно завершались позже, чем было запланировано.

Таблица D.7.
Работа по проверке соответствия требованиям с помощью различных типов тестов для различных CSCI

Тип теста	САС	СС	КВ	ТИЭ	ООД	ВВД	КТП	Всего
Версия 0/1 НТ	42			5				47
Версия 2 НТ	11	52	63	15	12			153
Версия 3/4/5 НТ		65	62	18	198	46		389
ФТ 1/2	131	39	77	94				341
ФТ 3	32	49	117	42				240
ФТ 4	16	172	219	5	4	6		422
ФТ 5/ОКТ	5	105	84	42	54	207	46	543
Всего	237	482	622	221	268	259	46	2135

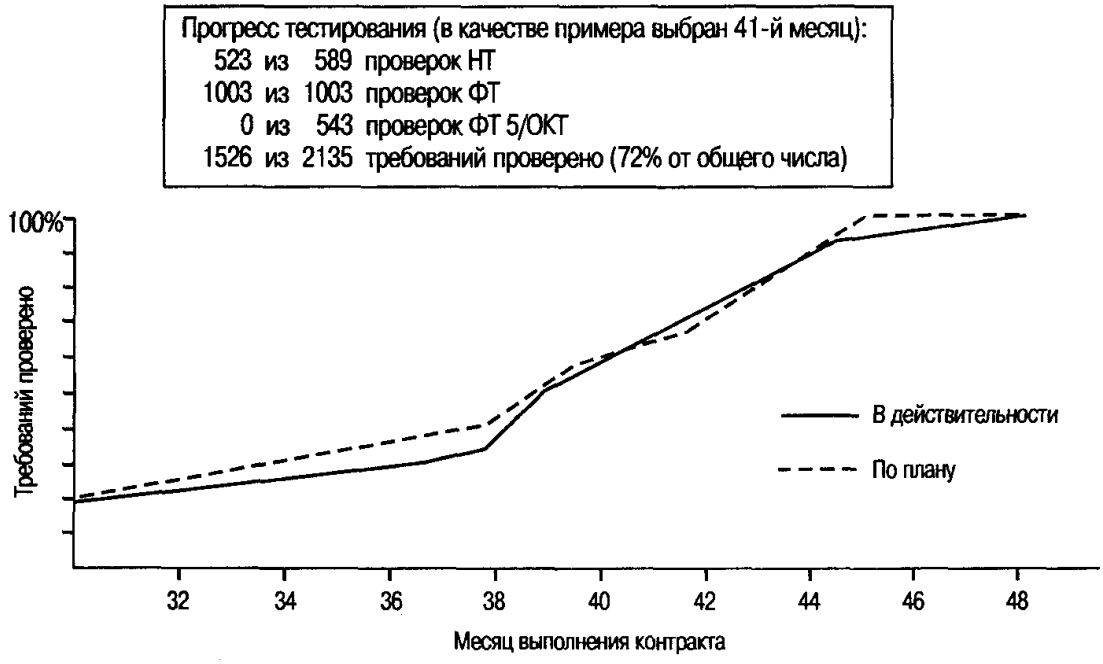


Рис. D.11. *Прогресс тестирования Подсистемы общего назначения*

D.7.3 Стабильность

На рис. D.12 приводится общий уровень изменений базовой конфигурации. Он показывает суммарное количество SLOC, признанных негодными (изъятых из базовой версии для доработки из-за обнаруженного дефекта, для расширения или для внесения другого вида изменений), и число восстановленных SLOC (тех, что были вновь включены в базовую версию с исправлениями, расширениями или какими-то другими изменениями). Скорость выявления дефектов, отличающаяся от скорости их исправления, приводила в результате к пристальному вниманию руководства, к изменению приоритетов при распределении ресурсов и к корректирующим мероприятиям, предпринимавшимся с целью убедиться в том, что отвечающая за тестирование организация (выявляющая дефекты) и организация-разработчик (выполняющая восстановление) находятся в относительном равновесии. В целом ситуация, изображенная на рисунке, относится к чрезвычайно благополучному проекту.

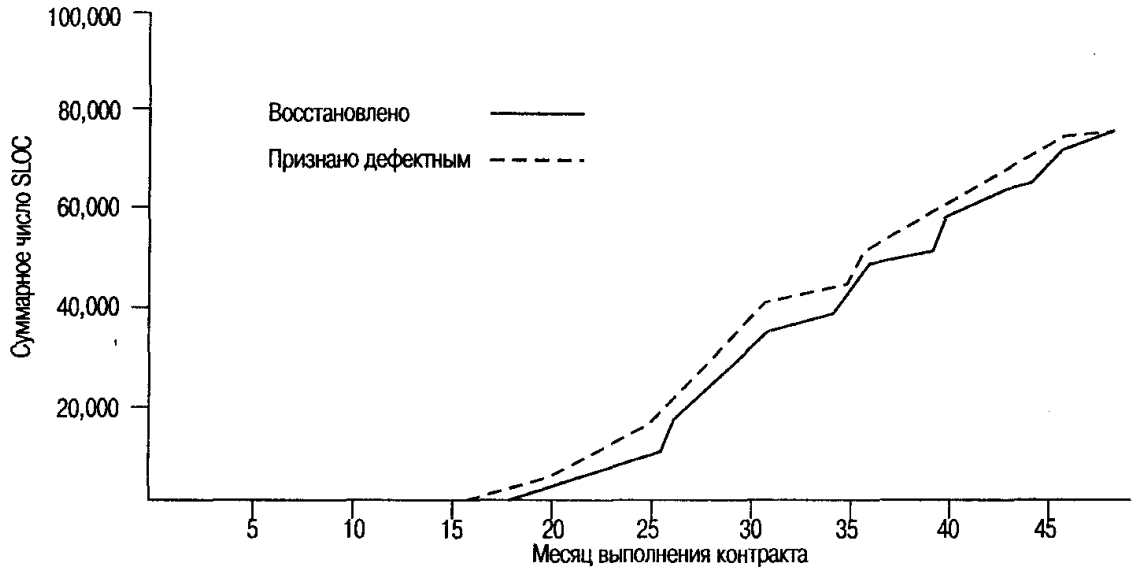


Рис. D.12. *Стабильность Подсистемы общего назначения*

D.7.4 Коэффициент дефектности

На рис. D.13 общее количество дефектов определяется относительно программной подсистемы в целом. Эта метрика оценивает суммарную дефектность, выявленную в процессе разработки Подсистемы общего назначения, приблизительно как 25% от объема всего продукта. В среднем в индустрии по созданию ПО средний объем дефектов колеблется в диапазоне от 40% до 60%. Начальная базовая конфигурация была создана к моменту ПОП, на 14 месяце. После этого в нее было внесено 1600 отдельных изменений.

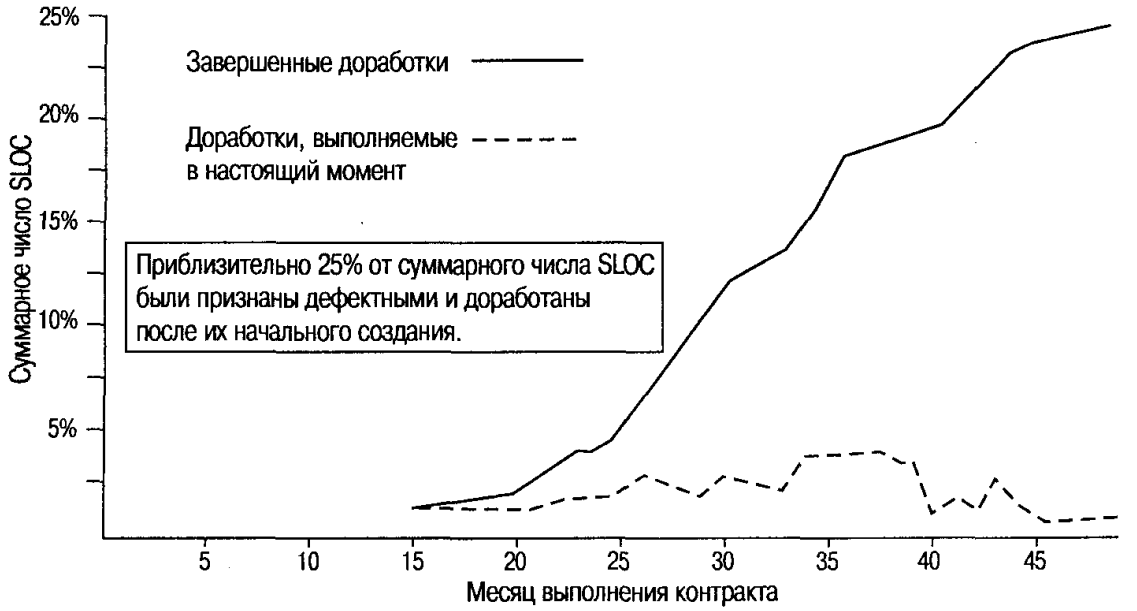


Рис. D.13. *Коэффициент дефектности в Подсистеме общего назначения*

D.7.5 Адаптируемость

Для Подсистемы общего назначения в целом на доработку базовой версии ПО было затрачено около 5% от всего объема работ. Средняя стоимость внесения одного изменения составляла около 24 ч на один SCO. Эти значения позволяют оценить ту легкость, с которой могли вноситься изменения в базовую версию ПО. Уровень адаптируемости, достигнутый в рамках проекта CCPDS-R, был примерно в четыре раза выше, чем для обычных проектов, в которых затраты на доработки на протяжении жизненного цикла обычно превышают 20% от общего уровня затрат.

На рис. D.14 показана средняя стоимость одного изменения в процессе создания Подсистемы общего назначения. К моменту ОКТ было обработано 1600 SCO, касающихся изменения основ конфигурации, что привело к стабильной стоимости одного изменения. Проект CCPDS-R оказался одним из немногих контрпримеров утверждения: «чем более поздние стадии жизненного цикла вы проходите, тем больше дорогостоящих проблем обнаруживаете».

Большинство SCO на ранних этапах (на рис. D.14 они изображены в прямоугольнике с надписью «Изменения в проекте») являлись изменениями, затрагивающими большое число сотрудников и большое количество компонентов (*изменения в интерфейсах и архитектуре*). *Более поздние* SCO (обозначены как «Изменения в реализации») обычно касались одного человека и одного компонента. Последний участок кривой отражает нетипичное возрастание дефектов, что стало результатом большого технического предложения о полном изменении набора входящих сообщений для Подсистемы общего назначения. Эта область являлась одной из тех областей, внесение изменений в которые было не таким простым делом, как хотелось бы. Хотя проект был устойчивым и приспособленным к большому числу предусмотренных заранее сценариев внесения изменений, пересмотр всего набора входных сообщений никогда не предполагался, да и проект не был для этого приспособлен.

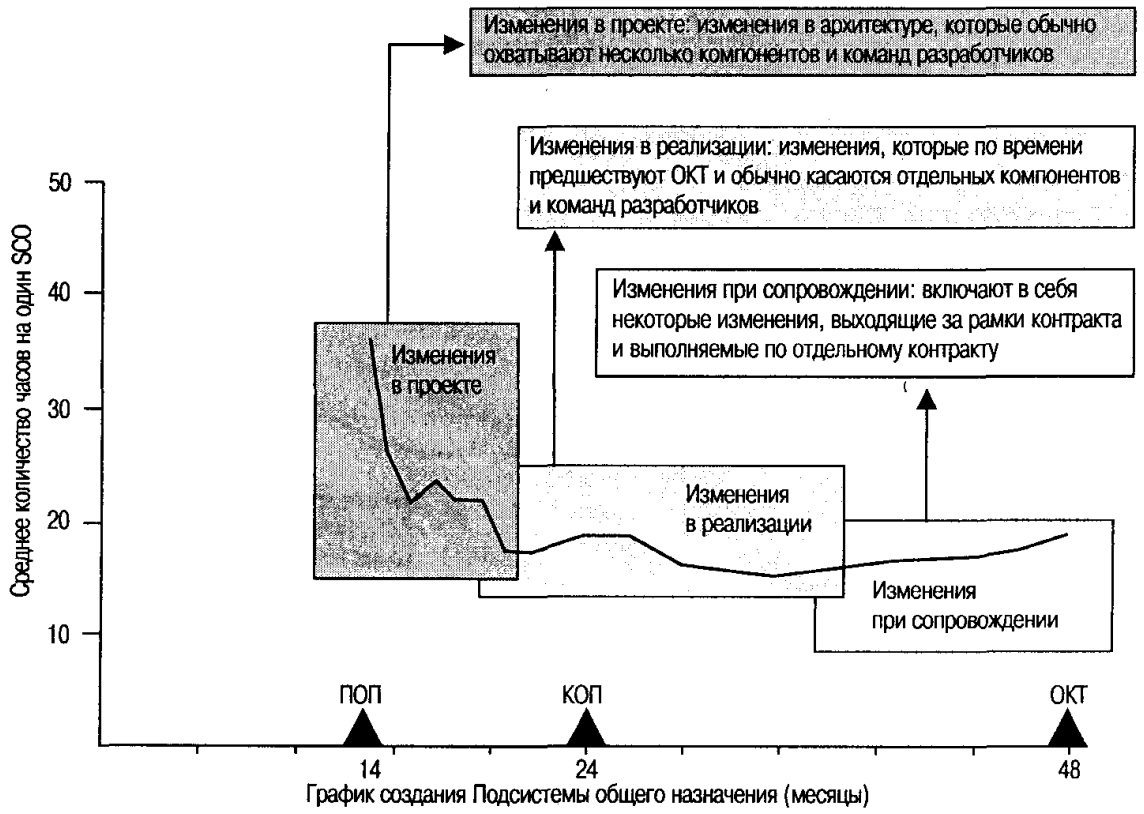


Рис. D.14. *Адаптируемость Подсистемы общего назначения*

D.7.6 Завершенность

К проекту CCPDS-R предъявлялись особенные требования по надежности, в связи с чем ПО было распределено особым образом. Выполняющая тестирование независимая команда создала автоматизированный набор тестов. Он проводился в неурочное время и испытывал базовую версию ПО по сценариям случайных сообщений. Такая стратегия привела к проведению широкого тестирования в условиях, близких к реальным на протяжении длительного времени. По результатам удалось определить значение MTBF для ПО. Критичные по надежности компоненты, принудительно перенесенные в плане итераций на самые ранние стадии, подвергались наиболее жесткому тестированию на надежность. Результаты показаны на рис. D.15.

Для современных распределенных архитектур такой способ статистического тестирования с одной стороны необходим для обеспечения максимального тестового покрытия, а с другой — полезен для обнаружения проблем, связанных с борьбой за ресурсы, тупиками, перегрузкой ресурсов, утечкой памяти и другими ошибками Гейзенберга. Выполнение случайных и ускоренных сценариев в течение длительных интервалов времени (на протяжении всей ночи или выходных) позволяет получить на ранних стадиях понимание общей целостности ресурсов системы.

Комплект тестов	Версии ПО	Продолжительность тестирования (часов)	Число критичных сбоев	Суммарное число сбоев
4	0,1,2,3,4,5	19 400	2	17
3	0,1,2,3,4	23 068	2	17
2	0,1,2,3,4	20 600	2	18
1	0,1,2	108 528	4	26

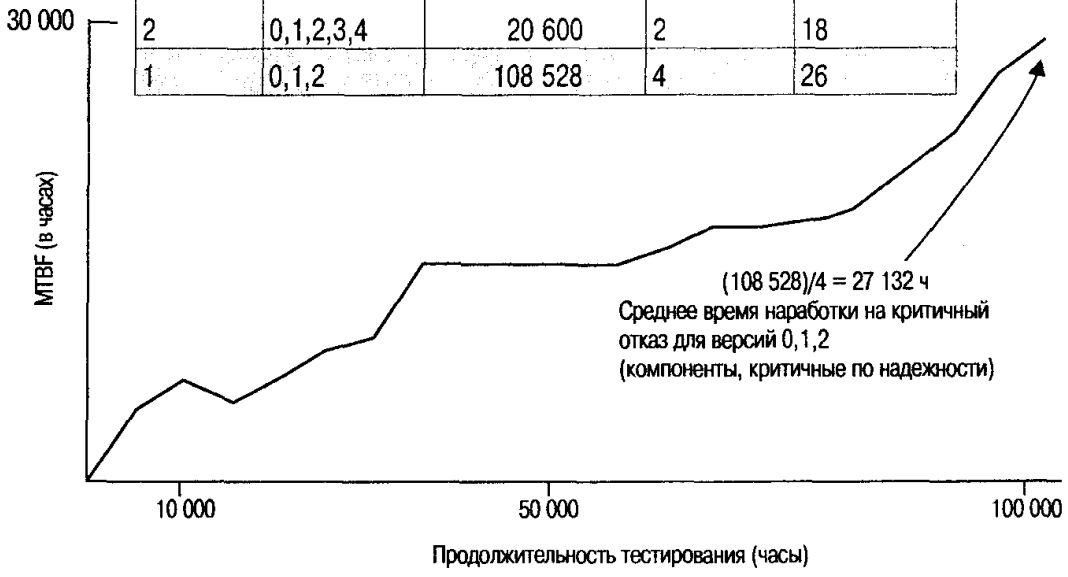


Рис. D.15. *Завершенность Подсистемы общего назначения*

D.7.7 Затраты финансов/работы на отдельные виды деятельности

В таблице D.8 рассматривается общая подробная структура затрат на Подсистему общего назначения в проекте CCPDS-R. Эти данные были получены из окончательного WBS-набора затрат и структурированы в соответствии с рекомендациями, приведенными в разделе 10.1. Элементы более низкого уровня описываются в таблице D.9.

Здесь необходимо сделать несколько замечаний:

- Проценты, указанные в таблице D.8, приблизительно соответствуют процентам, приведенным в главе 10. Однако некоторые из элементов таблицы D.9, касающихся управления, были распределены по нескольким элементам таблицы D.8 для выделения видов деятельности, находящихся на уровне управления проектом.
- Общие трудозатраты команды, осуществлявшей тестирование, оказались относительно низкими по сравнению с затратами в проектах, использующих традиционный процесс. Основная причина такого положения дел заключается в том, что команда по разработке архитектуры передавала интегрированный программный продукт команде, которая выполняла тестирование и оценку и отвечала прежде всего за тестирование интегрированного продукта.

Таблица D.8.
Финансовые затраты на Подсистему общего назначения по WBS-элементам самого верхнего уровня

WBS-элемент	Затраты (%)	Виды деятельности и рабочие продукты
Управление и администрирование	9	Окончательные планы, административная поддержка, финансовое администрирование, взаимодействие с заказчиком, контракты, общий контроль и руководство
Спецификация процесса/продукта	7	Технические требования, планы демонстраций и критерии оценки, планы итераций, процесс создания ПО, анализ метрик
Техническое обеспечение разработки ПО	11	Разработка архитектуры, координация сквозного контроля проекта, разработка CAC CSCI, определение и оценка метрик, планирование демонстраций и интеграция
Разработка	38	Разработка, тестирование, документирование и сопровождение компонентов приложений
Тестирование, оценка и внедрение	24	Управление версиями; подготовка, проведение и отчет о формальном тестировании; разработка сценария тестирования; управление изменениями; внедрение
Инфраструктура	11	Системное администрирование, программные и аппаратные ресурсы, настройка и интеграция инструментов
Всего работ по созданию ПО	100	Финансовые затраты, в том числе на аппаратные и программные средства (как элемент инфраструктуры), командировочные и другие прямые затраты

Таблица D.9.
WBS-элементы Подсистемы общего назначения более низкого уровня

WBS-элемент	Затраты (%)	Виды деятельности и рабочие продукты
Управление проектом по созданию ПО	6	Взаимодействие с заказчиком, контракты, администрирование
Техническое обеспечение разработки ПО	5	Координация требований, главный разработчик
Спецификации	4	Разработка СТПО CSCI
Демонстрации	3	Планы, интеграция, отчеты
Инструменты/метрики	3	Инструменты, сбор метрик
CAC CSCI	3	Промежуточное ПО, 20 KSLOC
Управление интеграцией и тестированием	4	Координация тестирования, управление
ИТВ-тестирование	3	Предварительное интеграционное тестирование

Таблица D.9. *(продолжение)*
WBS-элементы Подсистемы общего назначения более низкого уровня

WBS-элемент	Затраты (%)	Виды деятельности и рабочие продукты
ФТ-тестирование	9	Планы формального тестирования, тестирование, отчеты
ОКТ-тестирование	6	Планы формального тестирования, тестирование, отчеты
Управление конфигурацией и контроль среды тестирования	3	Управление версиями, интеграция
Среда	11	Аппаратные средства, программные средства, системное администрирование
Управление разработкой	5	Управление приложениями CSCI
СС CSCI	11	Архитектура, системное ПО, 160 KSLOC
КВ CSCI	9	Приложения интерфейса вывода, 70 KSLOC
ВВД CSCI	9	Коммуникационные приложения, 80 KSLOC
ТИЭ CSCI	2	Приложения для тестирования и испытания, 10 KSLOC
ООД CSCI	4	Приложения, содержащие алгоритм выполнения задания, 15 KSLOC
Всего работ по созданию ПО	100	Все затраты, связанные с созданием ПО

- В проекте CCPDS-R использовалась эффективная среда, на которую было затрачено 11% от общих ресурсов.
- Общее сопровождение (суммарные затраты на доработки) обошлось всего в 5% от общих затрат. Это не отражено явно в таблицах, но отслеживалось по отдельным CSCI WBS-элементам.

Чтобы сравнивать изменяющиеся уровни производительности абсолютно унифицированным способом, можно сопоставлять затраты на различные CSCI друг с другом так же, как и с другими метриками. При подобных сравнениях необходимо делать поправку на понимание руководством некоторых субъективных атрибутов, таких как конкуренция между командами, изменчивость требований, сложность конкретного CSCI и других факторов, не поддающихся сравнению.

> WBS самого верхнего уровня практически совпадает с рабочими процессами, описанными в разделе 8.1, и с распределением работ, рекомендованным в таблице 10.2.

D.8 ДРУГИЕ МЕТРИКИ

В этом разделе описывается более общий взгляд на выполнение проекта CCPDS-R: эволюция размера ПО, совершенствование процессов создания подсистем, диаграмма выполнения SCO, а также продуктивность различных CSCI и факторы качества.

D.8.1 Эволюция размера ПО

Размеры Подсистемы общего назначения и отдельных CSCI отслеживались ежемесячно и извлекались непосредственно из файлов с метриками. Объем кода существенно вырос: по сравнению с цифрой, указанной в контракте (150 000 SLOC), объем готового продукта составил 355 000 SLOC без сколь-нибудь существенного увеличения бюджета разработки ПО. Существуют два обоснования такого роста объема кода:

1. Способ подсчета количества исходных строк был изменен на восьмом месяце для того, чтобы улучшить баланс при оценке трудозатрат на разработку и не противоречить методу подсчета, принятому в модели Ada COCOMO.

2. Были разработаны автоматические инструменты для генерации кода, которые выдавали «многословный» исходный код при меньшем количестве входных строк, написанных людьми. Эти инструменты использовались для прямой генерации форматов вывода, обработки подтверждения сообщений и для функций учета взаимодействий. Сами инструменты состояли из 14 000 SLOC, и для них потребовалось 20 000 строк в файлах с входными данными. Выход от этих инструментов составил приблизительно 200 000 SLOC рабочего ПО. Таким образом, инструменты для генерации кода дали пятикратную отдачу от вложенного в них труда.

Общий рост объема кода приведен в таблице D.10.

Таблица D.10.
Размеры CSCI Подсистемы общего назначения

CSCI	Указано в контракте	Поставлено	Создано автоматически
CAC	20 000	20 000	
CC	18 000	160 000	140 000
KB	48 000	70 000	18 000
ТИЭ	17 000	10 000	4000
ООД	23 000	15 000	
ВВД	24 000	80 000	40 000
Итого	150 000	355 000	202 000

Главной причиной увеличения числа SLOC явилось изменение правил подсчета. При заключении контракта использовался простой подсчет символов «точка с запятой». Этот подход был перенесен в процедуру подсчета; она была реализована в виде простого инструмента, который использовался всем персоналом проекта:

- В рамках той части, которая определялась языком Ada, каждый символ «возврат каретки» считался за одну SLOC. Четыре стандарта кодирования обеспечивали непротиворечивость подсчета строк:

1. Каждый параметр в объявлении подпрограммы располагался на отдельной строке. Работа, связанная с написанием интерфейса подпрограммы, вообще говоря, прямо пропорциональна числу параметров.
2. Для специализированных перечислимых типов (например, наименований межпрограммных интерфейсов и состояний системы) и типов записей каждое перечисление или поле располагалось на отдельной строке. Специализированные типы обычно требуют специального проектирования и разработки, что приводит к увеличению числа SLOC.
3. Для предопределенных перечислимых типов (таких, как клавиши клавиатуры и стороны света) перечисление располагается на минимально возможном количестве строк без потери удобочитаемости. Такие типы обычно не требуют дополнительной разработки.
4. Инициализация сложных объектов (таких, как записи и массивы) располагается по одному компоненту на каждой строке. Каждое из этих присвоений представляется специально созданным выражением; служебное слово «others» (прочие) обычно используется для стандартных присвоений.

- Внутри подпрограмм, написанных на языке Ada, каждая точка с запятой принимается за одну SLOC. Общие описания считаются по одной строке на каждый общий параметр.

Такое определение более чувствительно к декларативной части проекта (спецификациям), чем к выполняемой части (телу). Оно вызвало множество жарких споров, тем не менее хорошо работает как внутри проекта, так и вне его. Не так важно иметь правильное определение; гораздо важнее, чтобы оно было непротиворечивым и адекватным.

Два компонента послужили причиной изменения определения SLOC. Во-первых, САПО-пакеты в СС содержали сетевые определения, в которые входили все определения процессов, задач, межпрограммных интерфейсов и соединений. В эти пакеты было включено множество определений записей, специально созданных перечислимых типов, а также инициализации полей записей и массивов в спецификациях. Исходный код для этих элементов содержал более 50 000 символов «возврат каретки», но всего лишь несколько сотен точек с запятой. Поскольку работа над этими пакетами больше была похожа на работу, связанную с написанием 50 000 SLOC, возникла необходимость в изменениях. Вторым компонентом с аналогичным обоснованием был компонент, содержащий типы глобальных сообщений системы. В этих пакетах содержалось около 300 различных типов записей, представляющих большую часть данных, которыми обменивались между собой САПО-объекты.

Из-за большого разнообразия категорий SLOC, разрабатывавшихся в рамках проекта CCPDS-R, был придуман метод нормализации различных категорий с тем, чтобы можно было правильно распределить бюджет и

сравнить продуктивность. В результате появилось расширение метода, предлагаемого моделью COCOMO для повторного использования; оно называлось эквивалентными строками исходного кода (ESLOC). По существу ESLOC преобразует стандартную для модели COCOMO единицу измерения SLOC в нормализованную единицу измерения, которая подлежит сравнению на основании работы, затраченной на написание одной строки кода. Необходимость в новой единице возникла при попытке распределения бюджета и анализа продуктивности для случаев, представляющих собой смесь из вновь разработанного, повторно используемого и автоматически сгенерированного кода. Например, компонент размером 10 000 SLOC, отвечающий за оформление вывода, который автоматически сгенерирован с помощью некоторого инструмента посредством задания 1000-строчного сценария оформления вывода, не должен иметь такого же бюджета, как и вновь разработанный компонент объемом в 10 000 строк. В таблице D.11 определяется перевод SLOC в ESLOC в проекте CCPDS-R.

Таблица D.11.
Факторы преобразования SLOC в ESLOC

Формат SLOC	Проектирование Новое = 40%	Реализация Новая = 20%	Тестирование Новое = 40%	ESLOC
Коммерческий	0%	0%	0%	0%
Новый	40%	20%	40%	100%
Повторно используемый	20%	5%	30%	55%
Автоматизированный	0%	0%	40%	40%
Входные данные для инструментов	30%	10%	10%	50%

При таких преобразованиях следует учитывать множество факторов:

- Готовые коммерческие компоненты не вносят никакого вклада при подсчете ESLOC. Включение таких компонентов учитывается с помощью количества вновь разрабатываемого ПО для реализации интерфейса с ними.
- Новое ПО разрабатывается с нуля. Для его создания требуется полный объем работ по проектированию, реализации и тестированию, поэтому ESLOC имеет множитель 100% (преобразование в соотношении один к одному).
- Код повторно используемых компонентов, которые были разработаны ранее, может применяться в другом компоненте при внесении некоторых изменений. Существует множество способов оценки относительных затрат на повторное использование, и для каждого конкретного случая лучше всего подходит свой собственный. Тем не менее такое преобразование определяется по умолчанию простым эмпирическим правилом. Вообще говоря, повторно используемый компонент требует 50% работ по проектированию,

25% по реализации и 75% по тестированию. Приведение в соответствие с распределением 40/20/40 для нового ПО дает итоговое значение 55%.

- Для автоматически генерируемых компонентов обычно требуется отдельная система учета исходных записей (формат входных данных для инструмента всегда меньше) как входных данных для инструмента, из которых в дальнейшем автоматически производятся конечные SLOC. Поскольку автоматически сгенерированный исходный код становится частью конечного продукта, он должен быть подвергнут исчерпывающему тестированию. А вот работа по проектированию и реализации равна нулю. Если инструмент, позволяющий автоматизировать создание исходного кода, должен быть разработан заново, его собственное число SLOC следует отнести к категории «новый». В результате получается значение фактора преобразования из SLOC в ESLOC, равное 40%.
- Входные данные для различных инструментов могут принимать самые разнообразные формы. В рамках проекта CCPDS-R использовались входные файлы для определения архитектуры (длинные, но простые таблицы имен, атрибутов и связей), определения вывода (типы выводимых объектов, местоположение и атрибуты) и подтверждения сообщений. Форматы с более высоким уровнем абстракции преобразовывались с затратами 75% работ на проектирование (простые нотации высокого уровня), 50% работ на реализацию (многократно повторяющиеся синтаксис и семантика высокого уровня) и 25% работ на тестирование (которые сосредоточивались на тестировании сгенерированного, а не исходного кода). В результате значение фактора преобразования из SLOC в ESLOC равен 50%.

Важно, что разработка нескольких инструментов для создания кода уменьшила значение ESLOC для Подсистемы общего назначения на 78 000 строк (см. таблицу D.12). Показатель ESLOC анализировался с единственной целью — убедиться в том, что общее распределение рабочей силы и бюджета, по которому велись переговоры с каждым руководителем CSCI, было относительно справедливым. Оценки в ESLOC использовались в качестве входных данных для анализа моделирования затрат, учитывающего относительную сложность каждого CSCI и другие уточняющие факторы модели COCOMO.

Изложенное на нескольких страницах, это описание подсчета строк кода может показаться весьма запутанным. Однако на протяжении первого года выполнения проекта этот анализ и определения подвергались тщательному изучению и были хорошо поняты. Они обеспечивали плодотворную точку зрения при обсуждении оценок некоторых соглашений, касающихся разработки. По истечении первого года подсчеты в SLOC стабилизировались и хорошо коррелировали с анализом оценки сроков, который проводился на протяжении всего жизненного цикла. С одной стороны, процесс подсчета строк кода, принятый в рамках проекта

CCPDS-R, хороший пример того, почему использование SLOC оказывается проблематичным при измерении размера ПО. С другой стороны, этот процесс является примером сложной системы, в которой единицы измерения SLOC работают весьма эффективно.

Таблица D.12.
Размеры CSCI Подсистемы общего назначения, выраженные в ESLOC

CSCI	Поставляемые SLOC	Сгенерированные с помощью инструментов	Использованные в качестве входных данных для инструментов	Разработанные инструменты	Размер (ESLOC)
CAC	20 000				20 000
CC	160 000	140 000	20 000	15 000	101 000
KB	70 000	18 000	6000	6000	68 800
ТИЭ	10 000	4000			7600
ООД	15 000				15 000
ВВД	80 000	40 000	12 000	3000	65 000
Итого	355 000	202 000	38 000	24 000	277 400

Данный раздел, посвященный размеру ПО, является отличным примером тех проблем, которые сопутствуют переходу к разработке, основанной на компонентах. Проекты могут и должны иметь дело с разнородными измерениями размеров, поскольку не существует подхода, общепринятого во всей индустрии. Из этого следует, что менеджерам проектов приходится тщательно анализировать определения этих важных метрик.

D.8.2 Совершенствование процессов создания подсистем

Одним из лейтмотивов данной книги является утверждение о том, что реальное улучшение процесса должно стать очевидным при последующем осуществлении проекта. Поскольку проект CCPDS-R состоит из трех отдельных проектов, он прекрасно иллюстрирует эту тенденцию. В целом создание Подсистемы общего назначения позволило выполнить основную черновую работу для подсистем PDS и STRATCOM, а именно: определение процесса, инструментария и архитектурных примитивов. При создании каждой очередной подсистемы продуктивность и качество значительно повышались. Это позволяло приближаться к зрелому процессу создания ПО, такому, каким является процесс, разработанный и усовершенствованный при осуществлении проекта CCPDS-R. Сравнивать производительность различных проектов всегда сложно, однако в случае

подсистем CCPDS-R имелись непротиворечивые способы измерения SLOC, созданных разработчиками, а также единые процессы, команды и методы. Согласованный подход к определению метрик обеспечивал получение сравнимых наборов данных. В качестве нормализованной единицы измерения были выбраны затраты, приходящиеся на одну SLOC. Абсолютные значения затрат несущественны; имеют значение лишь относительные затраты на каждую подсистему. Стоимость одной строки подсистемы PDS составляла 40% от стоимости SLOC Подсистемы общего назначения, а подсистемы STRATCOM – 33%. Это один из реальных показателей процесса уровня 3 или уровня 4 CMM.

В таблице D.13 представлено движение всех SCO по всем CSCI на 58-ом месяце выполнения проекта. К этому моменту для Подсистемы общего назначения закончился процесс ОКТ и было обработано некоторое количество SCO в режиме сопровождения с целью реализации некоторых новых предложений. Подсистемы PDS и STRATCOM уже находились в стадии тестирования. Для полноты картины в таблицу включены разделы, касающиеся вспомогательного ПО, тестирования и поставщика операционной системы. (Отслеживание запросов на внесение изменений в коммерческие продукты производилось аналогично отслеживанию SCO.) В вспомогательное ПО вошли инструменты для генерации кода, управления конфигурацией, работы с метриками и драйверы для независимого тестирования; к тестированию относились драйверы, использовавшиеся для проверок на соответствие требованиям.

Из таблицы D.13 видно, что значения коэффициента дефектности (среднее количество дефектов, приходящееся на одно изменение) и адаптируемости (среднее количество доработок на одно изменение) оказываются намного лучше для последующих подсистем (PDS и STRATCOM), нежели для Подсистемы общего назначения. Единственное исключение составляет CCB CSCI – специальная система взаимодействия, необходимая только для подсистемы STRATCOM и не имеющая аналогов в других подсистемах, поэтому ее сложность является уникальной.

Проект CCPDS-R продемонстрировал наличие реального показателя зрелости процесса в соответствии с описанным в разделе E.2. Для каждой последующей системы процесс ее создания — если измерять качество, продуктивность или затраченное время — оказывался лучше. За время жизни проекта CCPDS-R возможности по созданию ПО подвергались многочисленным оценкам, проводимым SEI, и зрелость процесса всегда соответствовала уровню 3 или выше. Однако эти улучшения возникли не только благодаря зрелости процесса. Работа в единой команде всех заинтересованных сторон и усилия, направленные на формирование основ архитектуры и автоматизацию процесса, оказались, вероятно, не менее важными для общего успеха проекта.

Таблица D.13.
Изменения, вносимые в подсистемы CCPDS-R по различным CSCI

CSCI	Общее количество SCO	Количество открытых SCO	Количество закрытых SCO	Количество отклоненных SCO	Средний дефект (SLOC/SCO)	Средний объем доработок (часов/SCO)
Подсистема общего назначения						
CAC	236	1	197	38	30	15
CC	1200	16	1004	180	24	16
KB	526	10	434	82	30	15
ТИЭ	255	0	217	38	40	11
ООД	123	2	105	16	24	35
ВВД	435	1	406	28	64	22
Подсистема PDS						
P CC	297	11	231	55	25	8
P KB	167	10	126	31	25	21
P ВВД	73	0	72	1	20	10
Подсистема STRATCOM						
S CC	531	30	401	100	18	10
S KB	339	11	286	42	16	14
S ТИЭ	60	0	50	10	20	9
S ООД	326	17	299	10	30	9
S ВВД	180	1	160	19	40	8
ССВ	61	6	51	4	85	27
Другие						
Вспомогательное ПО	648	2	546	100	Нет данных	Нет данных
Тестирование	376	1	356	19	Нет данных	Нет данных
Операционная система/ поставщик	223	13	161	49	Нет данных	Нет данных
Итого	6056	132	5102	822	32	13

D.8.3 Диаграмма выполнения SCO

Среднее значение затрат на внесение изменений со временем стремится к вполне постоянному значению — 16 ч на одно изменение. В эту работу включается время, потраченное на анализ, перепроектирование, повторное кодирование и повторное тестирование решения. Диаграмма внесенных изменений, представленная на рис. D.16, дает еще одну интересную точку зрения.

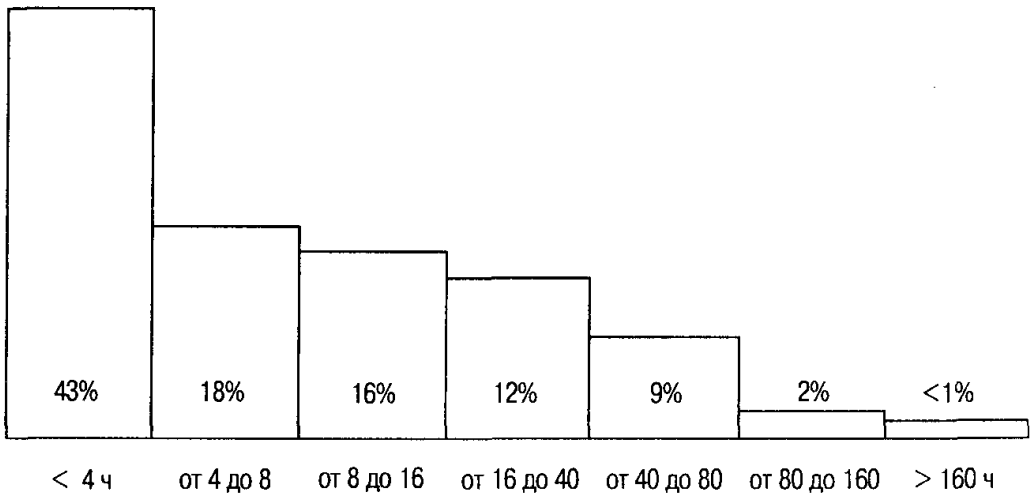


Рис. D.16. *Диаграмма изменения SCO для Подсистемы общего назначения*

D.8.4 Продуктивность и факторы качества для различных CSCI

В таблице D.14 представлены некоторые итоговые данные по продуктивности и качеству для различных CSCI в рамках проекта CCPDS-R. Значения продуктивности для различных CSCI не являются абсолютными; для того чтобы их можно было сравнивать между собой, они нормированы относительно общей продуктивности для подсистемы в целом. Продуктивность всей подсистемы базируется на суммарных затратах в размере приблизительно 1800 человеко-месяцев. Сюда входят ресурсы, потраченные на управление, разработку и тестирование. Значения продуктивности для каждого CSCI нормировались. Приводятся два различных значения продуктивности: SLOC на один человеко-месяц и ESLOC на один человеко-месяц. Эти данные и мой собственный опыт позволяют прийти к следующим заключениям:

- CAC оказалась чрезвычайно сложной проблемой в смысле разработки ПО, для решения которой требовались как высокая производительность, так и возможность повторного использования. Для ее создания была сформирована специальная команда. Разработка основывалась на существующем прототипе и имела адекватный график.
- У CC оказалось высокое абсолютное значение продуктивности, поскольку весь автоматически сгенерированный с помощью специально созданных CASE-средств код преимущественно содержался в данном CSCI. Команда, имевшая уровень выше среднего, также внесла свой вклад в улучшение продуктивности.

- КВ оказалась средней во всех отношениях, но в этом случае приходилось принимать во внимание высокую изменчивость требований, касающихся интерфейса вывода, без внесения соответствующих изменений в контракт. Создание этого CSCI и работа команды в данном случае были гораздо лучше, чем следует из приведенных цифр.
- ТИЭ имели самую низкую продуктивность, несмотря на самое простое и наиболее понятное ПО. Основная причина заключалась в том, что для этой команды выделялось меньших ресурсов, чем было запланировано для других команд. Еще одной причиной стало то, что команда, занимавшаяся ТИЭ, территориально находилась в другом месте и столкнулась с серьезными ограничениями по ресурсам среды разработки.
- ООД характеризовалась высокими затратами на внесение одного изменения и низкой продуктивностью без каких-либо видимых технических причин. Для гарантии сохранения технической целостности изменения, вносимые в алгоритм оповещения о запуске ракет, подвергались тщательному изучению многими заинтересованными сторонами. Необходимость в координации этого процесса приводила к высоким издержкам при улучшении продуктивности и при внесении изменений в ООД.

Таблица D.14.
Итоговые данные по различным CSCI для Подсистемы общего назначения

CSCI	Сложность	SLOC	Продуктивность (человеко-месяцы)		Дефекты (SLOC/SCO)	Доработка (часов/SCO)
			SLOC	ESLOC		
САС: сложное системное ПО	Очень высокая	20 000	260	260	30	15
СС: архитектура, ПО системы	Высокая	160 000	320	200	24	16
КВ: вывод, пользовательский интерфейс	Умеренная	70 000	170	160	30	15
ТИЭ: тестирование и эмуляция	Низкая	10 000	110	75	40	11
ООД: алгоритмы выполнения заданий	Умеренная	15 000	100	100	24	35
ВВД: внешние коммуникации	Высокая	80 000	170	140	64	22
Итого: подсистема оповещения *о запуске ракет*	Высокая	355 000	200	160	24	16

- У ВВД наихудшие качественные показатели. Прежде всего потому, что при разработке не была учтена возможность изменения всего комплекта основных сообщений, что привело к большому количеству трудноисправимых дефектов. Кроме того, команда разработчиков ВВД, возможно, хуже остальных (в общекультурном смысле) адаптировалась к процессу, метрикам и основанному на демонстрациях подходу, использовавшимся при выполнении проекта CCPDS-R.

В целом этот уровень продуктивности и качества превзошел стандарты компании TRW для предыдущих проектов по созданию ПО для командного центра приблизительно в два раза.

D.9 ЧЕЛОВЕЧЕСКИЙ ФАКТОР

В рамках проекта CCPDS-R для управления персоналом использовались два уникальных подхода. Первый — это принцип «ядра команды», при котором основное внимание уделялось эффективному использованию потенциала нескольких лучших экспертов команды. Второй был направлен на предотвращение увольнения сотрудников. CCPDS-R являлся первым большим Ada-проектом компании TRW, и менеджеры осознавали, что персонал, прошедший школу этого проекта, станет весьма привлекательным для выполнения других проектов как в компании TRW, так и где-либо еще. Для того чтобы материально заинтересовать сотрудников не покидать проект CCPDS-R в течение длительного времени, в рамках проекта была учреждена программа постепенного увеличения вознаграждения.

В результате применения общего подхода к управлению проектом CCPDS-R в процессе создания Подсистемы общего назначения текучесть кадров была очень маленькой, при этом большая часть команды разработчиков переходила к решению новых задач в заранее запланированных точках жизненного цикла. В противовес начальным ожиданиям работа по подсистемам PDS и STRATCOM так значительно перекрывалась с работами по Подсистеме общего назначения, что пришлось нанимать новых для данного проекта людей. Одним из примеров текучести кадров являлось возвращение сотрудников, составлявших ядро команды по разработке архитектуры (пятерых экспертов по CAC), к научно-изыскательским проектам, финансируемым самой компанией, по превращению CAC CSCI в коммерчески распространяемый системный продукт. Это произошло в момент достижения контрольной точки КОП.

D.9.1 Основная команда

Основная команда для разработки ПО в рамках проекта CCPDS-R была создана в самом начале стадии определения концепции специально для осуществления тех 20% процентов деятельности, касающейся разработки ПО, отдача от которых была наивысшей. В частности, в сферу ответственности этой команды числом не более десяти человек входило следующее:

1. Разработка наиболее мощных компонентов (преимущественно в рамках CAC CSCI). Эти компоненты позволяли решать многие сложные компьютерные проблемы, такие как распределение работ в режиме реального времени, взаимодействие между процессами, управление конфигурацией в процессе функционирования системы, обработка ошибок и программирование распределенных систем. В результате инкапсуляции всех этих сложных проблем в малое количество мощных компонентов основные компоненты стали намного проще и менее зависимы от высококвалифицированного персонала.
2. Установка стандартов и процедур сквозного контроля проекта и рабочих продуктов. Вообще говоря, основная команда находилась на переднем крае всех видов деятельности по созданию ПО. Практически во всех случаях эта команда первой выполняла каждый рабочий процесс в рамках проекта и создавала первую версию большинства рабочих продуктов. Соответственно, основная команда имела самое непосредственное отношение к созданию всех стандартов, будь это стандарты для какого-либо отдельного вида деятельности или формат/содержимое какого-либо конкретного вида рабочих продуктов.
3. Распространение культуры по всей организации-разработчику ПО. Основная команда действительно была единым тесно спаянным коллективом на протяжении начальной стадии и большей части стадии уточнения. По мере того как процесс и архитектура стабилизировались, члены команды начинали мигрировать, причем некоторые из них выдвигались на ведущие технические роли в различных командах, занимающихся разработкой и оценкой. В процессе создания и передачи некоторые из членов основной команды по-прежнему занимались тем, что следили за поддержанием архитектурной целостности всего проекта. Однако несколько глобально мыслящих индивидуумов, тесно связанных с командой, создающей архитектуру, с головой погрузились в другие области разработки и оценки. Такая команда и такие перемещения персонала оказались бесценным механизмом для поддержания общей культуры.

> Основная команда близка по своим задачам команде по созданию архитектуры, описанной в разделе 11.2.

D.9.2 План постепенного увеличения вознаграждения

В середине 1980-х гг. профессиональные знания в области ПО пользовались большим спросом. Бизнес компании TRW по созданию ПО и индустрия ПО в целом развивались быстрыми темпами. И менеджеры компании TRW, и правительственный заказчик были заинтересованы в том, чтобы набрать и сохранить стабильную квалифицированную команду программистов для проекта CCPDS-R. Для выполнения проекта также требовалось получение и совершенствование познаний в области языка Ada настолько, насколько это возможно, а знание языка Ada было весьма

дефицитным ресурсом на ранних стадиях проекта CCPDS-R. Компанией TRW был предложен новаторский подход к вопросу о распределении прибыли для увеличения возможностей по привлечению и удержанию дополнительных сотрудников в рамках проекта.

Основной предпосылкой плана постепенного повышения вознаграждения являлось то, что сотрудники должны были участвовать в дележе прибыли, полученной в результате выполнения проекта. (Вознаграждение — это предусмотренные контрактом дополнительные выплаты сверх основной суммы. Вознаграждения привязаны к ходу выполнения проекта, оцениваемому по заранее определенным критериям.) Руководство компании TRW решило распределять существенную часть общих вознаграждений по достижении каждой основной контрольной точки непосредственно среди тех, кто трудился над проектом. Эта дополнительная компенсация должна была распределяться среди отдельных сотрудников в соответствии с их относительным вкладом и сроком, в течение которого они работают над проектом. Реализация плана постепенного увеличения вознаграждения предназначалась для достижения следующих целей:

- Поощрение команды в целом за отличное выполнение проекта
- Поощрение различных отдельных групп пропорционально их вкладу в общее дело
- Поощрение ведущих исполнителей в каждой отдельной группе
- Минимизация случаев увольнения хороших сотрудников

Получившийся план оказался запутанным, но простым в реализации. В конце концов, этот план достиг цели, касающейся минимизации числа уволившихся, особенно на ранних стадиях, когда потеря ключевых работников могла стать опустошительной. Оглядываясь назад, можно заметить у этого плана один существенный недостаток, который заключается в том, что вознаграждения на ранних стадиях (на момент проведения ПОП и КОП) были менее значительными, чем вознаграждения на более поздних этапах. В результате команды, ответственные за стадии конструирования и ввода в действие, получили большие вознаграждения, чем команды, работавшие на начальной стадии и стадии уточнения.

Основная рабочая концепция этого плана заключалась в следующем:

- Руководство определяло отдельные группы (разработчики системы, разработчики ПО, бизнес-администрация и администрация).
- Каждые шесть месяцев все члены каждой группы расставляли самих себя по местам в соответствии с собственным вкладом в проект. Руководитель группы также проводил расстановку команды. Затем каждый руководитель сводил полученные расстановки в глобальный рейтинг работы своей группы.
- Каждый раз по достижении какой-либо контрольной точки сумма вознаграждения определялась заказчиком. Половина общей суммы вознаграждения при этом распределялась среди сотрудников, занятых в проекте.

- Алгоритм распределения вознаграждения среди сотрудников, заняты в проекте, был предельно прост. Общий уровень дополнительной компенсации по отношению к зарплате каждого сотрудника колебался в пределах 2% – 10% от суммы его ежегодной зарплаты.
  - Распределение вознаграждения между отдельными группами выполнялось в соответствии со средней зарплатой и числом работников в группе. Различия в зарплатах сотрудников внутри каждой группы как раз и определяли, что можно ожидать от данного сотрудника в смысле вклада в общий успех проекта.
  - Процесс распределения в каждой отдельной группе состоял из двух частей. Одна половина общей суммы вознаграждения, предназначенной для данной группы, распределялась поровну между всеми ее членами. Вторая половина распределялась между ведущими исполнителями в группе, которые определялись с помощью расстановки, выполняемой самими сотрудниками группы. Менеджерам предоставлялась некоторая свобода действий в определении сумм и процентов.

Трудно определить реальное влияние плана постепенного увеличения вознаграждения. Мне кажется, что он позволил выявить различия в общекомандной работе и способствовал удержанию наиболее ценных сотрудников. Групповые расстановки хорошо работали при определении ведущих исполнителей. Хотя нередко встречались некие сюрпризы, групповые расстановки довольно точно соответствовали расстановке, выполняемой руководством. Итоговые результаты проекта CCPDS-R говорят сами за себя. В целом компания TRW распределила среди сотрудников, работавших над проектом, чуть менее 10% своей общей прибыли. Проект CCPDS-R оказался весьма прибыльным для компании TRW и очень ценным для заказчика из военно-воздушных сил. Отдача от этих затрат расценивается всеми заинтересованными сторонами как очень высокая.

D.10 ЗАКЛЮЧЕНИЕ

Компания TRW и военно-воздушные силы тщательно документировали успех, достигнутый при использовании подхода с упреждающей разработкой архитектуры при выполнении проекта CCPDS-R. В ходе проекта удалось получить двукратное увеличение производительности и качества, при этом создание важных и ответственных систем уложилось в сроки и в рамки бюджета. Проект CCPDS-R оказался успешным в большой степени за счет взвешенного применения современных технологий, современного инструментария и итерационного процесса, очень похожего на процесс, описанный в данной книге. В таблице D.15 приведены многочисленные аспекты улучшений, внесенных в проект CCPDS-R. Достигнутая в результате эффективность может быть отнесена на счет существенного уменьшения дефектов и доработок ПО (почти на 25%), которое стало возможным благодаря тому, что архитектуре уделялось

первоочередное внимание, благодаря итерационному процессу, благодаря просвещенному и открытому для сотрудничества заказчику, а также благодаря применению современных среды, языков и инструментария.

В целом за счет Подсистемы общего назначения был выполнен большой объем черновой работы для подсистем PDS и STRATCOM, например: определение процесса, создание инструментария и повторно используемых примитивов архитектуры. Отдача от этих затрат в виде возросших продуктивности и качества была получена при создании последующих подсистем. Все это результат зрелого процесса по созданию ПО, такого, который был разработан и усовершенствован в рамках проекта CCPDS-R.

CCPDS-R был жестко регламентирован стандартом DOD-STD-2167A Министерства обороны, и в результате его выполнения были переданы все указанные в контракте документы по Подсистеме общего назначения. По мере того как заинтересованные стороны набирались опыта в осуществлении нового итерационного процесса и обзоров, основанных на демонстрациях, их давление на разработчиков с целью получения неэффективной документации начинало ослабевать. Заказчик и пользователь становились более осведомленными в вопросе расширения возможностей, чем это было при работе с бумагами.

Одним из самых главных (и тонких) улучшений, сделанных в результате применения такого подхода к разработке ПО в рамках CCPDS-R, явилась совместная работа заказчика, пользователя и подрядчика. Постоянный обмен информацией, переговоры и интерпретация результатов выполнения контракта оказались весьма продуктивными в смысле достижения реального прогресса и получения гарантий того, что каждая стадия жизненного цикла приводила к ситуации, полностью удовлетворяющей каждую заинтересованную сторону.

Уровень изменчивости требований был умеренным, но с бесконечными изменениями пользовательского интерфейса, алгоритмов оповещения о ракетных запусках и других аспектов, к которым приходилось приноравливаться в процессе выполнения проекта. Компании TRW также пришлось вносить изменения в архитектуру, в технологию и учитывать другие отклонения — касающиеся разработки — от первоначального технического задания. Требования постоянно менялись в процессе разработки и стабилизировались только после принятия КОП в качестве основы, подлежащей тестированию. Однако в конце выполнения проекта произошло еще одно существенное изменение рамок контракта, которое поменяло большую часть приведенных в настоящем практическом примере данных с целью получения более удобочитаемого представления. Эти изменения произошли на 35-ом месяце выполнения контракта, и касались они полного пересмотра формата входных сообщений для всей системы. Поскольку повышенное внимание уделялось производительности, большинство компонентов было создано с жесткой привязкой к формату входных сообщений. В отличие от других изменений архитектуры, алгоритмов и вывода такой вид изменений не был предусмотрен. Соответственно, в процессе разработки определенная легкость внесения изменений в формат сообщений была принесена в жертву для улучшения работоспособности. Дефект, явившийся следствием этого изменения, не

был настолько локализован, насколько этого хотелось, однако он был исправлен с предсказуемыми затратами. Поздний всплеск в объеме доработок (изменения при сопровождении, показанные на рис. D.14) явился результатом реализации этого существенного нововведения. Все заинтересованные стороны были удовлетворены окончательным решением.

Таблица D.15.
Улучшение технологии в рамках проекта CCPDS-R

Параметр	Современный процесс создания ПО	Подход в проекте CCPDS-R
Среда	Интегрированный инструментарий	Инструменты DEC/Rational/сделанные на заказ
	Открытые системы	Под управлением VAX/DEC
	Производительность аппаратуры	Несколько усовершенствований в семействе VAX
	Автоматизация	Специально разработанная система управления изменениями, инструменты для работы с метриками, аудиторы кода
Размер	Повторно используемые и коммерческие компоненты	Единые архитектурные примитивы, инструментарий, процессы, применявшиеся при создании всех подсистем
	Объектно-ориентированный подход	Основанная на сообщениях, объектно-ориентированная архитектура
	Языки высокого уровня	100% Ada
	CASE-средства	Специально созданные автоматические генераторы кода для архитектуры, ввода/вывода сообщений, вывода форматированного исходного кода
	Распределенное промежуточное ПО	Направление затрат на ранних стадиях на разработку CAC для обеспечения возможности повторного использования в различных подсистемах
Процесс	Итерационная разработка	Демонстрация, множественные версии, ранняя поставка заказчику
	Модели зрелости процессов	Процесс уровня 3 по определению SEI CMM
	Упреждающая разработка архитектуры	Выполняемая базовая архитектура к моменту ПОП
	Реорганизация процессов	Слаженная работа в единой команде заказчика/подрядчика/пользователя; отлично приспособленный к итерационной разработке стандарт 2167A
	Обучение	Обучение преимущественно в процессе работы и внутреннее наставничество

Описанная выше степень изменчивости требований привела бы к краху большинства проектов, использующих традиционный подход к управлению. Процесс выполнения проекта CCPDS-R поддерживался на должном уровне и при сохранении неантагонистических отношений между заинтересованными сторонами на протяжении всего жизненного цикла, сопровождающегося умеренным уровнем изменчивости требований. Хотя это трудно описать количественно и качественно, я думаю, что это является наиболее значительным достижением всего проекта.

> Как обсуждалось в главе 15, успешные проекты имеют тенденцию сохранять баланс между всеми необходимыми технологиями. Чрезмерное внимание, уделяемое какой-либо одной технологии, никогда не приводит к успеху. Взвешенный подход к технологиям необходим для успешного завершения большого проекта. В этом смысле проект CCPDS-R является хорошим примером. Были произведены значительные затраты на разработку правильного процесса, на интегрирование инструментария в эффективную среду и на разработку тех компонентов архитектуры, которые были необходимы для реализации подхода, основанного на демонстрациях. Все заинтересованные стороны (разработчики, менеджеры, заказчики и пользователи) были вовлечены в неантагонистические отношения и трудились во имя общих целей.

Команда проекта CCPDS-R была успешной на протяжении всего выполнения проекта. Многие люди и целые организации внесли свой вклад в успешное осуществление проекта, однако следующие специалисты оказали наибольшее влияние на общий подход к управлению проектом: Том Бостлаар, Чарльз Гролинг, Том Херман, Терри Крупп, Стив Петэй, Пэтти Шишидо и Майк Спрингмен (все из компании TRW); Джерри Лакруа (компания Mitre); Пол Харткист и Билл Веннингер (военно-воздушные силы США). Навыки управления проектами Дона Андреса (компания TRW) оказались очень важными для осуществления нового процесса по созданию ПО с огромным количеством здравых идей и для достижения успеха в боевых условиях широкомасштабного проекта государственной важности под пристальным вниманием со стороны множества правительственных организаций.

Приложение E

Усовершенствование процесса и соответствие CMM

Модель технологической зрелости (The Software Engineering Institute's Capability Maturity Model, SEI CMM) — широко известный способ оценки зрелости процесса создания ПО [SEI, 1993; 1993b; 1995]. CMM является популярным методом оценки зрелости процесса создания ПО для организаций во многих различных областях. Данное приложение предполагает базовое понимание CMM. Здесь проводится обсуждение текущего состояния CMM, как это обычно практикуется в промышленности. Основы определения зрелости процесса создания ПО описываются в книге «Managing the Software Process» [Humphrey, 1989].

Ключевые моменты

- Модель технологической зрелости (Capability Maturity Model, CMM) является отличной точкой зрения для оценки схемы процесса, представленного в данной книге. Надлежащим образом реализованный и используемый по убеждению, этот процесс может достигать 3 или 4 уровня зрелости.
- Реальным показателем зрелости процесса является предсказуемость его результатов и хода выполнения, что оказывает положительное влияние на последующие виды деятельности.
- Наличие зрелого процесса оказывается более важным, чем простое прохождение проверок.
- Зрелому процессу не страшны внезапные проверки. Если организация говорит, что она делает, и делает, что говорит, нет никакой необходимости специально готовиться к проверкам.

E.1 ОБЩИЙ ОБЗОР CMM

В CMM определены пять уровней зрелости процесса создания ПО, основанных на том, какие конкретные «ключевые» области процесса (Key Process Area, KPA) поддерживаются организацией. Уровень 1 (самый низший) соответствует организации с незрелым или неописанным

процессом. Уровень 2 (воспроизводимый), уровень 3 (определенный), уровень 4 (управляемый) и уровень 5 (оптимизированный) описывают организации с более высокими уровнями зрелости процесса создания ПО. KPA, соответствующую каждому из этих уровней, можно охарактеризовать следующим образом:

- KPA уровня 2: управление требованиями, планирование проекта по созданию ПО, отслеживание и контроль за выполнением проекта по созданию ПО, обеспечение качества ПО, управление конфигурацией ПО
- KPA уровня 3: большое внимание процессу организации, определение процесса организации, программа обучения, интегрированное управление созданием ПО, разработка программных продуктов, координация между группами, визуальные проверки
- KPA уровня 4: измерение и анализ параметров процесса, управление качеством, предотвращение появления дефектов
- KPA уровня 5: нововведения в технологию, управление изменением процесса

Целью большинства организаций является достижение процесса уровня 3. Для определения зрелости организации используется оценка возможностей ПО (Software Capability Evaluation, SCE). SCE позволяет выяснить, действительно ли организация «говорит то, что она делает, и делает то, что говорит», посредством оценки процесса создания ПО, используемого в данной организации (обычно в виде отдельных положений, определяющих политику), и практики выполнения проектов. Политика организации — «говорит, что она делает» — и реализация проектов — «делает то, что говорит» — оцениваются по соответствующей KPA-схеме. Процесс оценки не является совершенным, однако он может служить хорошим относительным показателем зрелости процесса создания ПО.

Для типичной SCE в качестве одной из частей тщательной проверки применяется SEI Maturity Questionnaire («Вопросник для определения зрелости») [SEI, 1998]. Это определение зрелости включает в себя подробный анализ, интервью и другие формы оценки. Данный вопросник, вообще говоря, используется в качестве отправной точки для создания контекста, в котором будет производиться оценка.

Существует большое число различных оценок распределения организаций, занимающихся созданием ПО, по этим пяти уровням. В таблице E.1 дается приблизительное распределение для индустрии ПО в 1995 г.

Одним из главных недостатков SEI CMM является то, что KPA ориентируются прежде всего на документальные рабочие продукты традиционного процесса (такие, как проект, требования и документы, определяющие соответствие требованиям), а также на контракты, контракты субподряда, планы и отчеты. Очень немногие KPA реально обращаются к изменяющимся рабочим продуктам, касающимся разработки (модели требований, проектные модели, исходный код или выполняемый код), к

уровню автоматизации процесса, обеспечиваемого средой, или к процессу создания архитектуры ПО. Другими словами, большая часть из моих 10 основных принципов современного процесса не учитывается на тех уровнях, которым она соответствует. Другим недостатком является присущее CMM рассмотрение управления конфигурацией и обеспечения качества как дисциплин, независимых от остальных видов деятельности в рамках процесса, а не как их неотъемлемой части.

Таблица Е.1.
Распределение по уровням зрелости в индустрии ПО

Уровень зрелости CMM	Частота	Основной уровень работы[1]
1 Начальный	70%	Непредсказуемый, с высоким риском
2 Воспроизводимый	15%	С трудом держатся на плаву, но выживают
3 Определенный	<10%	Стабильный, предсказуемый, прогрессирующий
4 Управляемый	<5%	Хорошо предсказуемый, надежный
5 Оптимизированный	<1%	Постоянно улучшающийся

На практике реальным показателем зрелости процесса является уровень предсказуемости хода выполнения проекта. Попытка поставить в соответствие ход выполнения проекта пяти уровням зрелости CMM может выглядеть следующим образом:

- Уровень 1 соответствует случайному (непредсказуемому) выполнению.
- Уровень 2 позволяет достигать воспроизводимого выполнения от проекта к проекту.
- Уровень 3 демонстрирует улучшение выполнения следующих друг за другом проектов в терминах затрат, сроков или качества.
- Уровень 4 свидетельствует о таком выполнении проектов, при котором для следующих друг за другом проектов происходит существенное улучшение либо одного из параметров выполнения процесса, либо нескольких параметров (например, затрат и качества).
- Уровень 5 соответствует идеальному выполнению следующих друг за другом проектов или существенному улучшению сразу всех параметров. Организации уровня 5 обычно занимают очень узкую нишу.

На рис. Е.1 показано ожидаемое выполнение следующих один за другим проектов в организациях с различным уровнем зрелости.

Многие организации могут делать вид, что их процесс соответствует уровню 3. Соответственно, процесс уровня 3 необязательно является хорошим процессом. С другой стороны, по-настоящему хороший процесс всегда с легкостью получит рейтинг, соответствующий уровню 3. Мой практический опыт, полученный при оценке десятков проектов по созданию ПО и определению возможностей ПО, дал мне понимание некоторых других показателей действительного процесса:

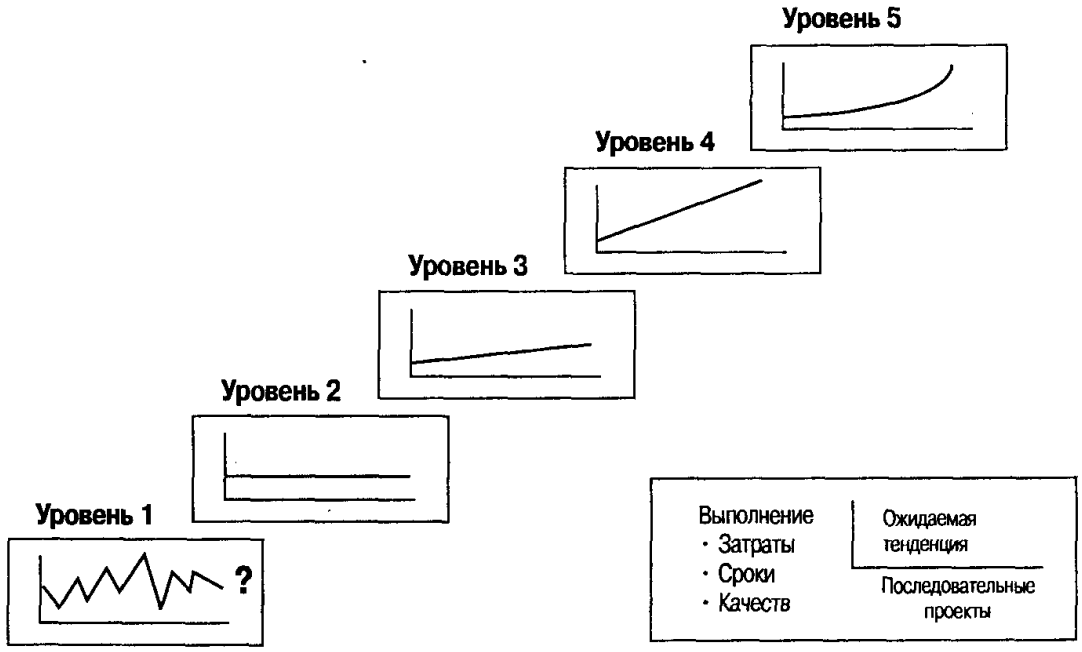


Рис. E.1. *Ожидаемое выполнение проектов для различных уровней зрелости CMM*

- Объективное понимание текущей зрелости
- Объективное понимание выполнения проекта в терминах затрат и сроков, поддающихся количественному выражению
- Реальное улучшение выполнения проекта
- Минимальное время, необходимое для подготовки к проведению оценки

Зрелая организация и зрелые проекты понимают процесс и следуют ему. Им не приходится тратить время на подготовку к проверке. Если вы думаете, что ваша организация соответствует уровню 3, то ответьте на такой вопрос: сможет ли она выдержать внезапную проверку?

E.2 ПРАКТИЧЕСКОЕ УЛУЧШЕНИЕ ПРОЦЕССА

В этом разделе приводятся некоторые предварительные размышления по общим вопросам улучшения процесса. Моей задачей является установить правильный баланс между надеждами и опасениями относительно тех перспектив, которые связаны с улучшением процесса.

- Зрелость процесса. Соответствие схемам определения качества процесса, например SEI CMM, необязательно приводит к получению качественного продукта. Однако действительно высококачественный процесс будет оценен как зрелый. Один из основных недостатков большинства схем процессов состоит в том, что они задают статически определенную программу подтверждения качества как чей-то независимый вид деятельности, вместо того чтобы динамически интегрировать обеспечение качества во все виды деятельности.

- Стоимость зрелого процесса. Зрелый процесс *не требует* большого количества денег. Напротив, он позволяет экономить деньги в долговременном смысле. Поскольку улучшение незрелого проекта всегда приводит к изменению структуры затрат, в организациях обычно обращается внимание на кратковременные затраты, связанные с улучшением процесса. Важным моментом здесь является то, что внушение мысли об улучшении процесса для тех проектов, которые уже находятся в стадии выполнения и в которых преобладает краткосрочный подход к определению затрат, является делом очень трудным. Однако необходимость в улучшении процесса легко внушить тем организациям, которые преследуют долговременные цели в бизнесе, и для тех долгосрочных проектов, которые все еще находятся в стадии планирования.

- Метрики ПО. Объективные измерения, необходимые для оценки качества программного продукта и прогресса в работе, — это два взгляда под различными углами зрения на создание ПО. Разработчики архитектуры больше интересуются показателями качества, в то время как менеджерам обычно нужны показатели прогресса. Успех любого процесса создания ПО, в котором значения метрик собираются вручную, будет ограниченным. Большинство самых важных метрик ПО — это простые, объективные измерения того, каким образом продукт/проект изменяется с различных точек зрения. Измерение абсолютных значений обычно менее важно, чем измерение относительных изменений в зависимости от времени. Из-за динамичного характера проектов по созданию ПО эти измерения должны быть доступны в любое время, необходима возможность их адаптации для различных составных частей постоянно изменяющегося продукта (подсистемы, версии, компонента, команды), и они должны содержаться в таком виде, чтобы можно было оценить тенденции (первую и вторую производные). Постоянная доступность была достигнута на практике только тогда, когда метрики стали поддерживаться в онлайновом режиме в качестве автоматизированного побочного продукта среды разработки.

- Адаптация процесса. Различные виды деятельности по созданию ПО требуют разных процессов. Имеются некоторые универсальные приемы и способы, но существуют также зависящие от ситуации различия в способах, приоритетах, церемонии и в том, на что обращается основное внимание. Разные ситуации при разработке ПО определяют различные требования, которые охватывают целый диапазон процессов. Внутренний процесс организации для создания некоторого продукта не будет в точности совпадать с процессом, который используется в проектах по разработке больших операционных систем по контракту со сторонним заказчиком.

- Процесс в сравнении с методом. Процесс управления проектом имеет дело с иными понятиями, нежели технический метод. Первый характеризуется итерационной разработкой, основанными на демонстрациях оценками и управлением рисками; второй – объектно-ориентированными методами, подходами к архитектуре и UML-представлениями. Плохое управление процессом, вероятно, никогда не сможет быть спасено с помощью хорошего метода, а вот хорошее управление процессом, скорее всего, приведет к успеху при использовании большинства технических методов. Понятно, что одни методы лучше других. Результат, который получается с помощью хорошего метода разработки, совмещенного с хорошим процессом управлением, – абсолютен. Это – основная цель.

E.3 ВОПРОСНИК ДЛЯ ОПРЕДЕЛЕНИЯ ЗРЕЛОСТИ

Ниже обсуждается подход к процессу, представленному в настоящей книге, с точки зрения SEI CMM. «Вопросник для определения зрелости» SEI [SEI, 1998] использовался мной в качестве сценария для оценки степени завершенности подхода к процессу с общепринятой точки зрения на зрелость процесса. Каждый цитируемый вопрос выделяется *курсивом*, а за ним следует мой ответ со ссылками на материалы, различные виды работ и контрольные точки процесса.

В некоторых ответах, например, касающихся вопросов обучения, схема процесса не предусматривает какого-либо конкретного подхода. Такие ответы являются специфичными для каждой организации; это означает, что для отдельной организации потребуется свой собственный механизм, определяемый ее внутренней практикой и культурой.

Управление требованиями, уровень 2

1. *Используются ли ваши системные требования, относящиеся к ПО, в качестве базы для разработки и управления проектом?*

   ▲ Требования к ПО содержатся в документе с общей концепцией и в модели вариантов использования. Каждой итерации соответствуют спецификации версии, в которых определяются цели для промежуточных контрольных точек. Все эти рабочие продукты входят в базовую версию ПО и подпадают под процесс управления изменениями.

2. *Вносятся ли необходимые уточнения в планы, рабочие материалы и виды деятельности по мере изменения требований, относящихся к ПО?*

   ▲ При итерационной разработке каждая новая итерация сопровождается новыми спецификациями версии и соответствующим обновлением технических рабочих продуктов. Назначением запросов на внесение изменений в ПО (SCO) типа 3 и является обращение к необходимым изменениям, вызванным изменением требований.

3. *Следует ли проект зафиксированной в письменном виде политике данной организации по управлению системными требованиями, относящимися к ПО?*

   ▲ Политика организации должна содержать в явном виде подход к определению и управлению всеми рабочими продуктами проекта, включая комплект рабочих продуктов требований.

4. *Обучался ли персонал, ответственный за управление требованиями, методам управления требованиями?*

   ▲ Обучение зависит от конкретной организации.

5. *Используются ли измерения, проводимые для определения состояния работ, при управлении требованиями (например, общее число изменений требований, которые предложены, открыты, приняты и внесены в базовую версию)?*

   ▲ SCO типа 3 должны отслеживаться и учитываться при периодических оценках состояния.

6. *Подвергается ли деятельность по управлению требованиями проверкам на подтверждение качества ПО (Software Quality Assurance, SQA)?*

   ▲ Контроль качества является обязанностью всех команд. Независимая организация, которая выполняет тестирование и на которую возлагается основная ответственность за обеспечение качества, просто не рассматривает вопросы управления требованиями; она активно участвует в создании спецификаций версии и в обеспечении ее соответствия набору требований. Совет по контролю за конфигурацией (Configuration Control Board, CCB) также рассматривает изменения требований, содержащиеся в SCO. Кроме того, комплект рабочих продуктов требований применяется при осуществлении разработки, связанной с усовершенствованием моделей вариантов использования, комплекта рабочих продуктов проектирования, комплекта рабочих продуктов реализации и при демонстрациях комплекта рабочих продуктов внедрения.

Планирование проекта по созданию ПО, уровень 2

1. *Документируются ли оценки (размера, стоимости и сроков) для использования при планировании и для контроля за ходом выполнения проекта?*

   ▲ WBS определяет базовую стоимость и план. Бизнес-план и план разработки ПО определяют приблизительный график, содержание итерации и приблизительные размеры с различных точек зрения. Оценки состояния являются механизмом, позволяющим отслеживать прогресс и качество по сравнению с базовыми планами и их уточнениями. На более низких уровнях в SCO отражаются подробные оценки, планы и реальное положение вещей.

2. *Содержатся ли в планах по созданию ПО работы, которые необходимо выполнить, и поручения, которые были даны в рамках проекта по созданию ПО?*

   ▲ Бизнес-план и план разработки ПО описывают работы самого верхнего уровня, которые необходимо выполнить, и они подписываются менеджером проекта как поручение. В WBS отражаются основные затраты и задачи для всех уровней управления. В SCO также содержатся виды работ и задачи более низкого уровня.

3. *Согласны ли все группы и лица, к которым это имеет отношение, со своими задачами, касающимися проекта по созданию ПО?*

   ▲ Декомпозиция работ (WBS) предоставляет механизм для проведения переговоров о поручениях между менеджером проекта и подчиненными ему менеджерами. SCO и CCB обеспечивают механизм для ведения переговоров о поручениях более низкого уровня.

4. *Следует ли проект политике организации при планировании работ по созданию ПО?*

   ▲ Политика должна обеспечивать организационную основу для планирования проектов. Инфраструктура организации должна также обеспечивать доступ к предшествующему опыту и к стандартным приемам планирования.

5. *Обеспечено ли планирование проекта по созданию ПО адекватными ресурсами (т.е. финансированием и опытным персоналом)?*

   ▲ Менеджер проекта по созданию ПО, ответственный за план, создает его и отвечает за его успех. Этот бизнес-план содержит ожидания и поручения, необходимые организации для определения отдачи от инвестиций по каждому виду работ. Адекватность ресурсов для планирования не определяется политикой. Хорошей отправной точкой может служить цифра 10% — приблизительно такая часть ресурсов проекта должна быть направлена на планирование и работы по управлению. Определение адекватных ресурсов специфично для каждого проекта. Эти оценки должны стать предметом тщательного изучения PRA и должны пересматриваться по достижении всех основных контрольных точек.

6. *Используются ли измерения для определения состояния работ по планированию проекта по созданию ПО (т.е. завершенности этапов работ по планированию)?*

   ▲ Метрики прогресса специально разработаны для того, чтобы обеспечить понимание критичных аспектов плана по сравнению с реальностью (прогресс разработки, прогресс тестирования, критерии оценки, соответствие которым уже достигнуто, реализованные сценарии, разработанные SLOC, число открытых SCO по сравнению с числом закрытых и т.д.).

7. *Проверяет ли менеджер проекта работы по планированию проекта по созданию ПО как на регулярной основе, так и при наступлении определенных событий?*

▲ Оценки состояния гарантируют, что менеджер проекта ответственен за периодические обращения к необходимым показателям управления и оценки рисков. Основные контрольные точки и описания версий являются аналогичным механизмом стимулирования для выполнения оценок при наступлении событий.

Отслеживание и контроль за выполнением проекта по созданию ПО, уровень 2

1. *Сравнимы ли реальные показатели выполнения проекта (сроки, размеры и затраты) с оценками, содержащимися в планах по созданию ПО?*

▲ Оценки состояния позволяют сравнивать планируемые результаты с реальными по показателям прогресса. Описания версий дают возможность сравнивать планируемые показатели качества (критерии оценки) с реально полученными результатами. В SCO также содержатся сравнения планируемых и реально полученных результатов для детального управления изменениями.

2. *Предпринимаются ли меры по корректировке в том случае, если реально полученные результаты значительно отличаются от планов проекта по созданию ПО?*

▲ Невыполненные критерии оценки обязательно должны упоминаться в описаниях версий и учитываться в последующих итерациях. Остальные отклонения от плана отражаются в оценках состояния, где доведение до конца требуется и контролируется.

3. *Согласовываются ли изменения в распределении обязанностей со всеми заинтересованными группами и лицами?*

▲ Изменения в распределении обязанностей обсуждаются при изменении WBS, планов разработки ПО и оценок состояния. Поручения более низкого уровня также рассматриваются ССВ, отслеживаются по SCO и отражаются в описаниях версий.

4. *Следует ли проект политике организации по отслеживанию и контролю за деятельностью по разработке ПО?*

▲ Политика организации должна описывать стандартный формат оценок состояния для конкретных наборов тем так, чтобы можно было сравнивать между собой различные проекты.

5. *Существует ли в рамках проекта лицо, на которое возложена особая ответственность за отслеживание продуктов и работ по созданию ПО (например, трудозатрат, сроков и финансирования)?*

▲ Таким ответственным лицом является менеджер проекта по созданию ПО. Оценки состояния предоставляют механизм, гарантирующий периодические рассмотрения и отчетность в соответствие с основой WBS.

6. *Используются ли измерения для определения состояния работ по отслеживанию и контролю за ПО (например, суммарных трудозатрат на выполнение работ по отслеживанию и контролю)?*

   ▲ Затраты, расписанные в WBS, и параметры прогресса обеспечивают механизм для отслеживания состояния работ и предоставляют инструменты и возможность общего контроля над всеми затратами на создание ПО.

7. *Рассматриваются ли работы по отслеживанию и контролю за ходом выполнения проекта вышестоящим руководством на регулярной основе (например, ход выполнения проекта, нерешенные проблемы, риски и необходимые действия)?*

   ▲ Это в точности соответствует назначению бизнес-плана (который обновляется при переходах от одной стадии жизненного цикла к другой), оценок состояния и рассмотрениям основных контрольных точек.

Управление субподрядчиками по созданию ПО, уровень 2

Заключение контрактов субподряда специально не оговаривается в схеме процесса, тем не менее предполагается, что все методы, инструменты и механизмы должны распространяться на всех субподрядчиков для того, чтобы процесс оставался однородным. Если это невозможно или если генеральный подрядчик не в состоянии четко разграничить работы, предназначенные для выполнения субподрядчиком, обладающим зрелым процессом, то привлечения субподрядчиков следует избегать. Для эффективного управления рисками необходимо контролировать количество и сложность внешних взаимодействий организации. Все решения о привлечении субподрядчиков должны быть документированы в бизнес-плане.

1. *Используется ли для выбора субподрядчиков документированная процедура, основанная на их способности к выполнению определенных видов работ?*

   ▲ Политика организации должна требовать от всего персонала, работающего над проектом, включая субподрядчиков, следования единому плану разработки. Для участия в выполнении проектов нужно привлекать субподрядчиков, о которых известно, что они имеют процесс, оценка зрелости которого не ниже зрелости процесса, применяемого в головной организации по выполнению проекта. (Другими словами, организация с уровнем 3 не должна привлекать субподрядчика с уровнем 2.)

2. *Вносятся ли изменения в контракты субподряда с обоюдного согласия основного подрядчика и субподрядчиков?*

   ▲ Здравый смысл подсказывает, что это всегда должно быть именно так.

3. *Производится ли регулярный обмен технической информацией с субподрядчиками?*

   ▲ Субподрядчики, следующие одному и тому же плану разработки, будут участвовать в одних и тех же обменах технической информацией, основных контрольных точках и оценках состояния.

4. *Отслеживаются ли результаты и ход выполнения работ по созданию ПО субподрядчиками в соответствии с тем, что им поручено?*

   ▲ Работа субподрядчиков, следующих одному и тому же плану разработки, должна контролироваться в соответствии с тем, что им поручено, точно так же, как это делается у генерального подрядчика.

5. *Следует ли проект политике организации по работе с субподрядчиками?*

   ▲ Документы, определяющие политику, требуют, чтобы субподрядчики использовали тот же процесс, который применяется генеральным подрядчиком.

6. *Обучаются ли лица, ответственные за управление субподрядом, управлять субподрядами по созданию ПО?*

   ▲ Обучение зависит от конкретной организации.

7. *Используются ли измерения состояния работ для управления субподрядом по созданию ПО (например, соответствие срокам с учетом дат планируемых поставок и трудозатрат, израсходованных на управление субподрядом)?*

   ▲ Субподрядчики должны управляться точно так же, как генеральный подрядчик.

8. *Проверяет ли менеджер проекта работы по выполнению субподряда по созданию ПО как на регулярной основе, так и при наступлении определенных событий?*

   ▲ Менеджер проекта по созданию ПО управляет субподрядчиками точно так же, как и всей остальной командой, работающей над проектом. Все решения, касающиеся поручений субподрядчикам, должны быть документированы в бизнес-плане, который обновляется при переходах от одной стадии жизненного цикла к другой.

Подтверждение качества ПО, уровень 2

Все виды деятельности и весь персонал вовлекаются в подтверждение качества ПО (SQA). Рекомендуется привлекать независимые команды для проведения оценки с той целью, чтобы работы по оценке качества, такие как тестирование и анализ метрик, могли выполняться параллельно (это дает эффективное использование времени) и независимо (для разнообразия технических точек зрения). Отчетность по вопросам качества распределена по различным командам в рамках организации. Однако для

того, чтобы можно было ответить на данные вопросы, работа, выполняемая независимой командой по оценке, должна быть максимально приближена к определению SQA, принятому в CMM.

1. *Планируются ли работы по SQA?*

   ▲ В плане разработки ПО описываются работы по тестированию, определению метрик и деятельность по контролю качества. В WBS содержатся многие детали плана. Спецификации версий также являются механизмом для планирования деятельности по SQA.

2. *Дает ли деятельность по SQA объективное подтверждение того, что продукты и работы по созданию ПО твердо следуют применяемым в данном случае стандартам, процедурам и требованиям?*

   ▲ Спецификации версии указывают цели конкретной итерации. Планы разработки ПО определяют стандарты и процедуры выполнения проекта. В спецификациях версии описывается также качество промежуточных продуктов относительно объективных критериев по принципу соответствует/не соответствует. CCB и SCO гарантируют, что процедуры и стандарты более низкого уровня проверяются и отслеживаются. На автоматизированные инструменты среды (компиляторы, создание документации, управление изменениями) должно быть возложено подтверждение того, что продукты точно соответствуют применяемым в данном случае стандартам.

3. *Передаются ли результаты SQA-рассмотрений и проверок заинтересованным группам и лицам (например, тем, кто выполнял работу, или тем, кто отвечает за ее выполнение)?*

   ▲ Все контрольные точки процесса и все виды работ, выполняемой CCB, являются SQA-рассмотрениями. Промежуточные результаты периодически документируются в описаниях версий. Все SCO рассматриваются CCB при участии заинтересованных групп.

4. *Рассматриваются ли случаи несоответствия стандартам, которые не решены в рамках проекта, вышестоящим руководством (например, отклонения от применяемых стандартов)?*

   ▲ Это является одной из прямых задач утверждения планов разработки и оценок состояния со стороны PRA (Project Review Authority). Регулярные PRA-рассмотрения касаются любых предполагаемых отклонений от политики организации по мере прогресса проекта.

5. *Следует ли проект политике организации по реализации SQA?*

   ▲ Политика организации и планы разработки ПО должны обеспечивать SQA-политикой организацию и проект соответственно.

6. *Обеспечивается ли выполнение работ по SQA адекватными ресурсами (например, финансированием и специально назначенным менеджером, который получает всю информацию о проблемах несоответствия ПО стандартам и принимает по ним решения)?*

▲ Адекватность ресурсов, выделяемых на SQA, не определяется политикой. Хорошей контрольной цифрой являются 25% от всех работ в рамках проекта, направляемых на работу команды по выполнению оценок (тестирование, оценка, метрики, ССВ). Несмотря на то, что адекватность ресурсов специфична для каждого конкретного проекта, эти оценки, очевидно, должны выполняться под тщательным контролем PRA.

7. *Используются ли измерения для определения затрат и сроков по работам, связанным с SQA (например, объем завершенных работ, трудозатраты и финансовые затраты по сравнению с запланированными)?*

▲ Основа содержится в WBS, а периодические оценки состояния позволяют отслеживать реальность по сравнению с планами для всех видов деятельности.

8. *Рассматриваются ли работы по SQA совместно с вышестоящим руководством на регулярной основе?*

▲ Это одна из основных задач PRA-рассмотрений и PRA-утверждения плана разработки ПО.

Управление конфигурацией ПО, уровень 2

Все виды деятельности и весь персонал вовлекаются в управление конфигурацией ПО (Software Configuration Management, SCM) точно так же, как в SQA. Независимая команда, выполняющая оценку, принимает на себя основную ответственность за деятельность по управлению конфигурацией, включая сопровождение базы данных SCO, администрирование ССВ и управление базовой версией. Эти виды работ должны выполняться параллельно (для экономии времени) и независимо (для разнообразия технических точек зрения). Вообще говоря, действия по SCM выполняются всеми разработчиками ПО и поддерживаются прежде всего средой разработки ПО. Для ответов на эти вопросы деятельность независимой команды по оценке максимально приближена к определению SCM, принятому в CMM.

1. *Были ли запланированы работы по управлению конфигурацией в рамках проекта?*

▲ План разработки ПО автоматически должен содержать и поддерживать работы по SCM.

2. *Являлось ли управление конфигурацией средством идентификации, контроля и поставки рабочих продуктов, полученных в результате создания ПО?*

▲ SCO и все рабочие продукты находятся под управлением конфигурацией.

3. *Следует ли проект зафиксированной процедуре контроля над изменениями, вносимыми в элементы/модули конфигурации?*

▲ SCO обеспечивают механизм контроля над изменениями в онлайновом режиме, как это записано в политике организации и в плане разработки ПО.

4. *Распространяются ли среди заинтересованных групп и лиц стандартные отчеты по базовым версиям ПО (например, протоколы заседания ССВ, обзор запросов на внесение изменений, отчеты о состоянии)?*

▲ Оценки состояния, которые содержат в себе результаты работы ССВ в виде стандартных параметров, должны быть доступны для всех заинтересованных сторон и команд проекта.

5. *Следует ли проект политике организации в области реализации работ по управлению конфигурацией ПО?*

▲ Это обеспечивается политикой организации и планами разработки ПО.

6. *Обучается ли персонал выполнению тех видов деятельности по управлению конфигурацией ПО, за которую он отвечает?*

▲ Обучение зависит от конкретной организации. Вообще говоря, каждый работающий в организации, занимающейся созданием ПО, сталкивается с управлением конфигурацией в той мере, в какой это определяется средой. Формальные виды деятельности по управлению конфигурацией независимой группы, отвечающей за тестирование, прежде всего касаются административного контроля и составления отчетов.

7. *Используются ли измерения для определения состояния работ по управлению конфигурацией ПО (например, затраты труда и финансов на работы по управлению конфигурацией ПО)?*

▲ WBS обеспечивает базу, а периодические оценки состояния позволяют определить реальный ход всех работ по сравнению с запланированным.

8. *Производятся ли регулярные проверки на соответствие базовых версий ПО и их документации (например, SCM-группой)?*

▲ Наиболее частые проверки согласованности базовых версий ПО на основании SCO выполняются ССВ. Описания версий являются проверками общего качества, завершенности и непротиворечивости промежуточных базовых версий, созданных по достижении основных контрольных точек.

Наиболее важные аспекты процесса, используемого в организации, уровень 3

1. *Координируется ли в рамках организации деятельность по разработке и совершенствованию процесса создания ПО (например, специальной группой по разработке процесса создания ПО)?*

▲ Все планы по разработке и оценке состояния рассматриваются и утверждаются органом, ответственным за процесс разработки ПО (Software Engineering Process Authority, SEPA). Эти механизмы обеспечивают координацию и непротиворечивость в рамках организации.

2. *Регулярно ли оценивается процесс создания ПО в вашей организации?*

   ▲ SEPA является ответственным за регулярное проведение оценок, таких как анализ тенденций. Планирование этих оценок и определение их количества должны описываться в приложении к политике организации.

3. *Следует ли ваша организация документированному плану разработки и усовершенствования своего процесса создания ПО?*

   ▲ SEPA должен документировать этот план и приложить его к политике организации.

4. *Участвует ли вышестоящее руководство в деятельности, осуществляемой в рамках организации, по разработке и усовершенствованию процесса создания ПО (например, утверждая долгосрочные планы и направляя ресурсы и финансирование)?*

   ▲ Степень участия вышестоящего руководства в этой деятельности легко определить по составу группы SEPA и по деталям политики организации. Другим показателем участия руководства является та степень, с которой процесс, используемый в данной организации, поддерживается посредством капиталовложений в автоматизацию.

5. *Несет ли отдельное лицо или группа лиц полную или частичную ответственность за процесс создания ПО, используемый в данной организации (например, группа по разработке процесса создания ПО)?*

   ▲ Эта ответственность возлагается на SEPA. Состоит ли данный орган из одного человека, работающего по совместительству, или это команда, занимающаяся только этим вопросом, зависит от специфики организации.

6. *Применяются ли измерения для определения состояния деятельности по разработке и усовершенствованию процесса создания ПО, используемого в данной организации (например, объем трудозатрат на оценку и усовершенствование процесса создания ПО)?*

   ▲ Политика организации должна учитывать ROI для SEPA-деятельности и периодически обновляться.

7. *Регулярно ли деятельность по разработке и совершенствованию процесса создания ПО рассматривается вышестоящим руководством?*

   ▲ Следует обращаться к генеральному менеджеру организации для утверждения всех текущих изменений в политике организации.

Определение процесса, используемого организацией, уровень 3

1. *Разработан и поддерживается ли вашей организацией стандартный процесс создания ПО?*

   ▲ Политика организации определяет стандарт процесса создания ПО, который периодически обновляется.

2. *Занимается ли организация сбором, рассмотрением и обеспечением доступа к информации, касающейся вопросов использования стандартного процесса создания ПО (например, оценок и реальных данных по размерам, трудозатратам и финансовым затратам; данных по производительности; измерений, касающихся качества)?*

   ▲ SEPA принимает участие в проведении всех оценок состояния и содержит архив со всей информацией, включая результаты проведенных оценок, планы разработки ПО, данные по выполнению предыдущих проектов, а также другие стандартные инструменты и компоненты организации.

3. *Следует ли организация заданной политике в области разработки и поддержки своего стандартного процесса создания ПО?*

   ▲ Политика организации определяет стандарт процесса и его поддержку.

4. *Получают ли сотрудники, которые разрабатывают и поддерживают стандартный процесс создания ПО, используемый в данной организации, необходимую для осуществления этой деятельности подготовку?*

   ▲ Обучение зависит от конкретной организации.

5. *Применяются ли измерения для определения состояния деятельности, направленной на описание и поддержку стандартного процесса создания ПО, используемого в данной организации (например, состояния планируемых контрольных точек и затрат на деятельность по определению процесса)?*

   ▲ Информация о ROI от деятельности SEPA должна храниться в виде приложения к политике организации.

6. *Подвергаются ли деятельность и промежуточные продукты, связанные с разработкой и поддержкой стандартного процесса создания ПО, используемого в данной организации, SQA-рассмотрениям и проверкам?*

   ▲ Деятельность и промежуточные продукты постоянно рассматриваются практиками. Генеральному менеджеру организации следует по мере необходимости собирать соответствующий контролирующий орган с целью проверки того, что дополнительное обеспечение процесса, используемого организацией, является адекватным. SEPA является высшим органом внутри данной организации, ответственным за SQA. Он прекращает все ненужные разговоры до вмешательства генерального менеджера.

Программа обучения, уровень 3

1. *Планируется ли деятельность по обучению?*

   ▲ Обучение зависит от конкретной организации.

2. ***Направленно** ли обучение на совершенствование навыков и знаний, необходимых для того, чтобы занимать управленческие и технические посты?*

▲ Обучение зависит от конкретной организации.

3. ***Проходят*** *ли члены групп по разработке и других групп, имеющих отношение к созданию ПО, обучение, необходимое для выполнения своих обязанностей?*

▲ Обучение зависит от конкретной организации.

4. *Следует ли ваша организация заданной политике для удовлетворения ее собственных нужд в обучении?*

▲ Обучение зависит от конкретной организации.

5. *Выделяются ли достаточные ресурсы на реализацию программы обучения, принятой в организации (например, финансирование, программный инструментарий, соответствующие условия для обучения)?*

▲ Обучение зависит от конкретной организации.

6. *Используются ли измерения для определения качества программы обучения?*

▲ Обучение зависит от конкретной организации.

7. *Проверяется ли вышестоящим руководством деятельность, связанная с программой обучения, на регулярной основе?*

▲ Обучение зависит от конкретной организации.

Управление интегрированным ПО, уровень 3

1. *Был ли описанный процесс создания ПО, использованный для выполнения* ***проекта****, разработан путем адаптации стандартного процесса создания ПО, применяемого внутри* ***организации****?*

▲ Политика организации определяет обязательные механизмы, точку отсчета и степени свободы для создания процесса данного проекта. Процесс, используемый в проекте, контролируется и утверждается SEPA.

2. *Осуществляется ли планирование и управление процессом в соответствии с описанным для проекта процессом создания ПО?*

▲ Менеджер проекта по созданию ПО составляет и утверждает план разработки ПО, он же является ответственным за план при периодических PRA-обзорах.

3. *Следует ли проект заданной политике организации, требующей, чтобы он планировался и управлялся с использованием стандартного для данной организации процесса создания ПО?*

▲ Политика организации — это политика, зафиксированная в виде документов.

4. *Требуется ли обучение тем сотрудникам, перед которыми поставлена задача приспособить стандартный для данной организации процесс создания ПО к описанию процесса создания ПО для нового проекта?*

   ▲ Решение проблемы обучения зависит от конкретной организации. Составлять план разработки ПО должен менеджер проекта по созданию ПО.

5. *Используются ли измерения для определения эффективности деятельности по управлению интегрированным ПО (например, частота, причины и размеры затрат на изменение планов)?*

   ▲ В проектах используются оценки состояния, включая необходимые метрики, для оценки прогресса и качества. Эти метрики собираются и анализируются SEPA и PRA для определения эффективности и любых требуемых усовершенствований.

6. *Подвергается ли деятельность по управлению проектом и промежуточные продукты, получаемые в результате ее осуществления, SQA-рассмотрению и проверкам?*

   ▲ План разработки ПО рассматривается как PRA, так и SEPA (который отвечает за SQA в данной организации).

Разработка программного продукта, уровень 3

1. *Создаются ли промежуточные программные продукты в соответствии с определенным для данного проекта процессом создания ПО?*

   ▲ Менеджер проекта по созданию ПО ответственен за выполнение плана разработки ПО. Любые отклонения от плана или от стандартов (или от того и другого одновременно) периодически рассматриваются при оценке состояния, после чего вносятся соответствующие изменения в последующие итерации или базовые версии продуктов.

2. *Поддерживается ли согласованность промежуточных программных продуктов (например, отражается ли в документации соответствие требований к системе текущим требованиям к ПО, проекту, коду и вариантами тестирования)?*

   ▲ CCB уделяет постоянное внимание управлению изменениями. Описание версии является механизмом, позволяющим оценивать согласованность и завершенность промежуточных продуктов по достижении основной контрольной точки. Соответствие между комплектами рабочих продуктов разработки (моделями вариантов использования, проектными моделями, исходным кодом и выполняемыми компонентами) поддерживается за счет среды. Степень, с которой информация обобщается или детализируется, зависит от масштаба проекта и требований заинтересованных сторон (например, по вопросам безопасности) и отражается в описаниях версий.

3. *Следует ли проект заданной политике организации при осуществлении деятельности по разработке ПО (например, политике, которая предполагает использование конкретных методов и инструментов для создания и сопровождения программных продуктов)?*

   ▲ Политика организации требует определенных видов деятельности и использования некоторой стандартной среды с целью стандартизации методов или инструментов, применяемых в различных проектах. Вопрос об использовании остальных методов и инструментов остается открытым и решается по-своему в каждом конкретном проекте.

4. *Обеспечивается ли выполнение заданий по разработке ПО адекватными ресурсами (например, финансированием, обученным персоналом и необходимым инструментарием)?*

   ▲ Адекватность ресурсов, выделяемых на разработку ПО, не определяется политикой. Хорошей контрольной отметкой может служить цифра 50% — именно такую часть всех трудозатрат следует направлять на выполнение задач по разработке ПО: 10% — на описание требований, 15% — на проектирование и 25% — на реализацию компонентов. Определение адекватности ресурсов и персонала специфично для каждого конкретного проекта и должно тщательно изучаться PRA.

5. *Используются ли измерения для определения функциональных возможностей и качества программных продуктов (например, количество, типы и серьезность обнаруженных дефектов)?*

   ▲ Основная цель вычисления метрик и их рассмотрения при выполнении оценок состояния заключается в определении прогресса и качества.

6. *Подвергаются ли деятельность и промежуточные продукты деятельности по разработке ПО SQA-рассмотрениям и проверкам (например, выполняется ли необходимое тестирование, определяется ли соответствие требований к системе требованиям к ПО, проекту, коду и вариантам тестирования)?*

   ▲ Все комплекты технических рабочих продуктов изменяются и обновляются при достижении каждой основной контрольной точки. SCO, CCB и описания версий способствуют привлечению постоянного внимания к наличию такого соответствия.

Координация между группами, уровень 3

Этот набор вопросов пристальное внимание уделяет архитектуре. Координация между группами тесно связана с архитектурой ПО, поскольку архитектура охватывает вопросы взаимодействия между компонентами и между людьми.

1. *Сотрудничают ли группа по разработке ПО и другие группы разработчиков в процессе выполнения проекта с заказчиком для определения системных требований?*

▲ Общая концепция проекта и спецификации версий входят в сферу ответственности группы по разработке архитектуры ПО. Они обговариваются с заказчиком и изменяются в каждой итерации.

2. *Согласны ли группы разработчиков с возложенными на них обязанностями, которые зафиксированы в общем плане проекта?*

   ▲ Планы и поручения содержатся в планах разработки ПО, спецификациях версий и WBS.

3. *Занимаются ли группы разработчиков идентификацией, отслеживанием и разрешением проблем, которые касаются сразу нескольких групп (например, несоответствие графиков, технические риски или проблемы уровня системы в целом)?*

   ▲ Демонстрации являются тем механизмом, который обеспечивает продуктивную и реальную координацию на уровне архитектуры. CCB решает проблемы, касающиеся нескольких групп одновременно, на уровне SCO. Составление правильного графика демонстраций архитектуры позволяет решать проблемы, возникающие при интеграции, на максимально ранних стадиях жизненного цикла. Правильный график также расширяет возможности по разрешению на ранних стадиях важных проблем, касающихся сразу нескольких групп.

4. *Существует ли политика организации, касающаяся создания междисциплинарных групп разработчиков?*

   ▲ CCB, PRA и команды по проведению демонстраций как раз и являются междисциплинарными командами.

5. *Позволяют ли вспомогательные инструменты, используемые различными группами разработчиков, осуществлять эффективное взаимодействие и координацию (например, совместимые текстовые процессоры, базы данных и системы отслеживания проблем)?*

   ▲ Стандартная WBS, стандартная среда и база данных SCO позволяют различным группам разработчиков осуществлять координацию в рамках единой схемы. В рамках одного проекта рабочие продукты, создаваемые всеми командами, должны использовать одинаковые нотации, методы и инструменты.

6. *Используются ли измерения для определения состояния деятельности по координации между группами (например, трудозатрат на поддержку одной группой разработчиков ПО других групп)?*

   ▲ Вычисление трудозатрат команды по разработке архитектуры позволяет определить стабильность архитектуры. Стабильность архитектуры является хорошим показателем эффективности межгрупповой координации. Поскольку команда по разработке архитектуры является независимой, с четко определенными WBS-элементами, контроль за ее деятельностью проще всего осуществлять с помощью заранее определенного управления и отчетов по метрикам качества при периодических оценках состояния.

7. *Рассматривается ли деятельность по координации между группами менеджером проекта на регулярной основе и при наступлении определенных событий?*

   ▲ При правильном подходе с упреждающей разработкой архитектуры предполагается, что любые значимые проблемы в межгрупповой координации проявятся на начальных итерациях. Периодические оценки состояния и достижение основных контрольных точек позволяют получать конкретное и объективное понимание состояния межгрупповой координации посредством наблюдения за параметрами архитектуры.

Экспертные оценки, уровень 3

Схема процесса специально не предполагает наличия экспертных оценок в классическом смысле. Однако существуют некоторые механизмы, цель которых в точности совпадает с классическими экспертными оценками. В число этих механизмов входят демонстрации (экспертные оценки глобальной интеграции), ССВ (экспертные оценки управления изменениями), оценки состояния (экспертные оценки управления) и традиционные экспертные оценки (сквозной контроль кода, инспекции) — все они включаются в планы разработки ПО в рамках данного проекта.

1. *Запланированы ли экспертные оценки?*

   ▲ ССВ, оценки состояния и демонстрации следует включать в планы и проводить систематически.

2. *Отслеживается ли деятельность по устранению дефектов, обнаруженных при экспертных оценках, до тех пор, пока они не будут исправлены?*

   ▲ Исправление всех дефектов — независимо от того, каким именно способом они были обнаружены, — отслеживается посредством SCO, а все метрики определяются для оценок состояния.

3. *Следует ли проект политике организации в области выполнения экспертных оценок?*

   ▲ Политика организации должна предопределять наличие ССВ, демонстраций и оценок состояния. В ней также должно быть указано, что другие формы экспертных оценок описываются в планах разработки ПО в рамках конкретного проекта.

4. *Проходят ли участники экспертных оценок необходимое обучение для выполнения своих обязанностей?*

   ▲ Обучение зависит от конкретной организации.

5. *Используются ли измерения для определения состояния деятельности по проведению экспертных оценок (например, количество выполненных оценок, вложенные в экспертные оценки трудозатраты и число промежуточных продуктов, подвергнутых оценкам, по сравнению с планом)?*

▲ CCB предоставляют широкий набор метрик для управления изменениями. Для описаний версий требуется сбор тех же метрик, что и для демонстраций. SEPA периодически оценивает ROI анализа тенденций для данной организации посредством данных оценки состояния.

6. *Подвергаются ли деятельность и промежуточные продукты, связанные с экспертными оценками, SQA-рассмотрениям и проверкам (например, проводятся ли планируемые просмотры, отслеживаются ли окончательные проверки)?*

▲ Менеджеры проектов по созданию ПО, CCB и PRA постоянно следят, чтобы эти виды деятельности доводились до конца.

Управление количественными показателями процесса, уровень 4

1. *Следует ли проект документированному плану управления количественными показателями процесса?*

▲ В приложении к политике организации должен определяться план улучшения количественных показателей процесса. Оценки состояния позволяют оценить собранные метрики в контексте тех норм для данной организации, которые устанавливаются SEPA.

2. *Контролируется ли ход описанного процесса по созданию ПО в рамках данного* ***проекта*** *количественно (например, посредством использования количественных методов анализа)?*

▲ Все данные собираются и передаются SEPA в процессе выполнения оценок состояния. Эти метрики служат для установления обратной связи при планировании каждой последующей итерации.

3. *Известны ли количественные характеристики возможностей стандартного процесса по созданию ПО в рамках данного проекта?*

▲ В приложении к политике организации должны быть описаны процедура оценки текущего состояния процесса и план улучшения процесса в терминах количественных показателей.

4. *Следует ли проект политике организации в области измерения и контроля за ходом выполнения описанного процесса по созданию ПО в рамках данного проекта (например, как в рамках проекта планируется обнаруживать, определять и контролировать особые причины отклонений)?*

▲ План разработки ПО должен определять метрики для измерения и контроля за процессом создания ПО. В нем также должно содержаться требование, чтобы этот процесс (и механизмы его контроля) изменялись и совершенствовались по ходу выполнения проекта.

5. *Адекватны ли ресурсы, выделяемые на осуществление деятельности по управлению количественными показателями процесса (например, финансирование, вспомогательные программные инструменты и программа выполнения измерений для данной организации)?*

   ▲ Адекватность ресурсов, выделяемых на управление процессом, не определяется политикой. Хорошей контрольной цифрой является следующее значение: достаточное число команд приблизительно равняется корню квадратному из числа выполняемых проектов.

6. *Используются ли измерения для определения состояния деятельности по управлению количественными показателями процесса (например, затраты на управление количественными показателями процесса и на достижение контрольных точек при управлении количественными показателями процесса)?*

   ▲ В приложении к политике организации должна быть ссылка на ROI от SEPA-деятельности.

7. *Проверяет ли менеджер проекта деятельность по управлению количественными показателями проекта как на регулярной основе, так и при наступлении определенных событий?*

   ▲ Выполнение оценок состояния и основные контрольные точки предопределяют как регулярные, так и при наступлении определенных событий проверки данных по управлению количественными показателями проекта.

Управление качеством ПО, уровень 4

1. *Запланирована ли в рамках проекта деятельность по управлению качеством ПО?*

   ▲ Спецификации версий определяют ожидаемые значения параметров качества. Эти спецификации создаются для каждой итерации и документируются в оценках состояния (свидетельствах прогресса) и в описаниях версий (для основных контрольных точек).

2. *Ставятся ли в проекте определенные количественно и по приоритетам цели по управлению качеством его программных продуктов (например, функциональные возможности, надежность, удобство сопровождения и эксплуатации)?*

   ▲ Для этого существуют спецификации версий и соответствующие им демонстрации, описания версий и метрики.

3. *Позволяет ли сравнение измерений качества программного продукта с целевым качеством программного продукта определить, достигнуты ли цели?*

▲ Общая концепция проекта, спецификации версий и описания версий позволяют отслеживать достижение целевого качества. Выполнение оценок состояния также регулярно дает информацию относительно достижения определенного минимального набора показателей качества.

4. *Следует ли проект политике организации в области управления качеством ПО?*

   ▲ Политика организации действительно определяет механизмы управления качеством ПО (спецификации версий, описания версий, параметры качества, PRA и CCB/SCO).

5. *Проходят ли члены группы по разработке ПО и других групп, имеющих отношение к ПО, необходимое для управления качеством ПО обучение (например, обучение по сбору измеряемых данных и по преимуществам количественного определения качества продукта)?*

   ▲ Обучение зависит от конкретной организации.

6. *Используются ли измерения для определения состояния деятельности по управлению качеством ПО (например, затраты на низкое качество)?*

   ▲ Метрики дефектов и доработок являются достоверными показателями хода процесса в направлении достижения качества и затрат, связанных с доработками, вызванными неадекватным качеством промежуточных продуктов.

7. *Проверяется ли вышестоящим руководством деятельность по управлению качеством ПО на регулярной основе?*

   ▲ Деятельность и параметры, касающиеся качества, должны регулярно проверяться вышестоящим руководством.

Предотвращение дефектов, уровень 5

1. *Планируется ли деятельность по предотвращению дефектов?*

   ▲ Вопрос не рассматривается.

2. *Предусматривается ли в проекте проведение встреч для анализа причин, в ходе которых можно было бы установить наиболее распространенные причины дефектов?*

   ▲ Вопрос не рассматривается.

3. *После выявления причин дефектов расставляются ли наиболее распространенные причины по значимости и производится ли их систематическое устранение?*

   ▲ Вопрос не рассматривается.

4. *Следует ли проект политике организации в области деятельности по предотвращению дефектов?*

▲ Вопрос не рассматривается.

5. *Проходят ли члены группы по разработке ПО и других имеющих отношение к ПО групп необходимое для осуществления их деятельности по предотвращению дефектов обучение (например, обучение по методам предотвращения дефектов, по проведению встреч до начала выполнения задач и по анализу причин появления дефектов)?*

▲ Обучение зависит от конкретной организации.

6. *Проводятся ли измерения деятельности по предотвращению дефектов (например, временных и финансовых затрат на обнаружение и исправление дефектов, а также количества предложенных, открытых и завершенных проблем)?*

▲ В проектах должен использоваться набор метрик, позволяющий определять эффективность исправления и состояние дефектов. Деятельность по предотвращению дефектов специально не описывается и не подвергается формализации, но ее основы четко определены посредством параметров управления и качества. PRA-рассмотрения, SCO и CCB предоставляют различные механизмы для предотвращения дефектов на практике.

7. *Подвергаются ли деятельность по предотвращению дефектов и ее промежуточные продукты SQA-рассмотрениям и проверкам?*

▲ Все оценки состояния и PRA-рассмотрения предоставляются SEPA (для SQA данной организации). Оценки состояния распространяются среди персонала проекта и заинтересованных сторон, а данные по SCO рассматриваются CCB для подтверждения того, что весь персонал вносит свой вклад в получение данных по предотвращению дефектов.

Управление изменением технологии, уровень 5

1. *Следует ли организация плану управления изменениями технологии?*

▲ Вопрос не рассматривается.

2. *Оцениваются ли новые технологии с точки зрения того эффекта, который они оказывают на качество и производительность?*

▲ Вопрос не рассматривается.

3. *Следует ли* ***организация*** *документированной процедуре включения новых технологий в свой стандарт процесса по созданию ПО?*

▲ Вопрос не рассматривается.

4. *Способствует ли вышестоящее руководство управлению изменением технологии (например, создавая долговременные планы и поручения по финансированию, по найму персонала и выделению других ресурсов)?*

▲ Вопрос не рассматривается.

5. *Имеются ли данные по процессу, помогающие в выборе новой технологии?*

   ▲ Вопрос не рассматривается.

6. *Используются ли измерения для определения состояния деятельности в рамках управления изменением технологии (например, эффекта от реализации изменений в технологии)?*

   ▲ Вопрос не рассматривается.

7. *Подвергается ли деятельность в рамках управления изменением технологии рассмотрению вышестоящим руководством на регулярной основе?*

   ▲ Вопрос не рассматривается.

Управление изменением процесса, уровень 5

1. *Следует ли организация документированной процедуре разработки и сопровождения планов по усовершенствованию процесса создания ПО?*

   ▲ В политике организации должно иметься приложение, касающееся усовершенствований процесса.

2. *Участвуют ли сотрудники вашей организации в деятельности по совершенствованию процесса создания ПО (например, в командах по разработке усовершенствований процесса создания ПО)?*

   ▲ Участвуют SEPA и PRA. Участие остальных зависит от конкретной организации.

3. *Постоянно ли вносятся изменения в стандарт процесса по созданию ПО данной организации и в процесс создания ПО для данного проекта?*

   ▲ На каждой новой итерации при осуществлении проекта имеется возможность внесения усовершенствований в план создания ПО. Политика организации также нуждается в периодической переоценке и усовершенствовании.

4. *Следует ли организация принятой политике в области реализации усовершенствований процесса создания ПО?*

   ▲ Такая политика должна включаться в виде приложения в общую политику организации.

5. *Требуется ли обучение усовершенствованиям процесса создания ПО для руководящего состава и для технических работников?*

   ▲ Обучение зависит от конкретной организации.

6. *Используются ли измерения для определения состояния деятельности по совершенствованию процесса создания ПО (например, эффекта от реализации каждого усовершенствования процесса по сравнению с объявленными целями)?*

▲ От политики организации и SEPA требуются регулярные оценки ROI от деятельности по усовершенствованию в рамках данной организации.

7. *Рассматривается ли деятельность по совершенствованию процесса создания ПО вышестоящим руководством на регулярной основе?*

▲ Генеральный менеджер организации отвечает за утверждение всех изменений в политике организации.

E.4 ВОПРОСЫ, КОТОРЫЕ НЕ ВОШЛИ В «ВОПРОСНИК ДЛЯ ОПРЕДЕЛЕНИЯ ЗРЕЛОСТИ»

Для оценки зрелости процесса данной организации я бы задал еще несколько групп вопросов, которые на сегодняшний момент не являются ключевыми областями для CMM. В то время как предыдущие ответы непосредственно следуют «Вопроснику для определения зрелости», приводимые ниже вопросы соответствуют другим политике, механизмам и подходам к современному процессу, для которых CMM является слабой мотивацией. Эти вопросы помогут при оценке дополнительных критериев, определяющих успех схемы современного процесса.

Отчетность персонала, уровень 2

Одной из важных задач управления созданием ПО являются четкое распределение обязанностей и механизмы отчетности. Соответственно, приводимые ниже вопросы также следовало бы задать для оценки реализации процесса в данной организации.

1. Вводит ли организация должность для сопровождения описания своего процесса и других активов?
2. Хранится ли у SEPA история проектов и их воздействия на корпоративную стратегию?
3. Составляют ли менеджеры проектов по созданию ПО свои планы разработки?
4. Представляют ли менеджеры проектов по созданию ПО оценки состояния самостоятельно?

Автоматизация среды, уровень 3

Организация, в которой отсутствует требуемый уровень автоматизации, вряд ли будет обладать зрелым процессом. Ответы на следующие вопросы помогут оценить уровень автоматизации данной организации.

1. Описывает ли политика организации минимальное ядро среды?
2. Поддерживает ли среда набор метрик в качестве естественного побочного продукта процесса?

3. Является ли документация автоматизированным однородным побочным продуктом разработки (в отличие от отдельного, неоднородного вида материалов)?
4. Автоматизирована ли «круговая» разработка?
5. Можно ли сказать, что рабочие продукты тестирования создаются и сопровождаются с помощью тех же самых инструментов, методов и управления изменениями, что и основные рабочие продукты?
6. Адекватно ли автоматизировано регрессионное тестирование?
7. Используются ли случайные сценарии и внеурочное тестирование для обеспечения статистического тестового покрытия?
8. Интегрированы ли редакторы языков программирования, среда управления конфигурацией, компилятор и отладчик?

Разработка архитектуры, уровень 3

Организация, в которой не уделяется систематически большое внимание архитектуре, вряд ли станет обладательницей зрелого процесса. Ответы на следующие вопросы помогут выполнить оценку с этой точки зрения.

1. Доступно ли инструментальное оснащение, которое обеспечивало бы объективное понимание характеристик демонстраций (в отличие от разрабатываемого заново для каждого проекта)?
2. Уделяется ли основное внимание при экспертных оценках действительно критичным компонентам?
3. Существует ли четко сформулированное определение архитектуры?
4. Существует ли строгая проектная нотация, из которой можно получить объективную оценку прогресса проекта, его изменчивости и качества проектных решений?
5. Обладает ли команда по разработке архитектуры достаточными знаниями и опытом как в прикладной области знаний, так в программной инженерии?

Управление изменениями, уровень 3

В случае итерационного процесса разработки управление изменениями и простота внесения изменений являются ключевыми показателями зрелости процесса.

1. Имеют ли вносимые в ПО изменения однозначное определение их типа и приоритета?
2. Собираются ли, оформляются и анализируются данные по внесенным изменениям?
3. Определены ли и распределены обязанности по сопровождению базы данных изменений?

4. Существует ли механизм подтверждения того, что инициатор запроса на внесение конкретного изменения удовлетворен реализацией данного изменения?
5. Существуют ли учет вносимых изменений и демонстрируемая возможность отказа от внесенных изменений для целей управления рисками?

Предсказуемость, уровень 3

Самым лучшим показателем зрелости процесса является предсказуемость получаемых результатов.

1. Производит ли организация сравнение запланированных и реально полученных данных?
2. Имеется ли документально зафиксированная корреляция между планировавшимися и реально полученными данными по последним выполненным проектам?
3. Используются ли такие «исторические» планы и реальные данные в текущих проектах?
4. Ведутся ли документальные записи действий в случаях, когда реально полученные данные отличаются от запланированных?

Интегрированная экономика, уровень 4

Соответствующая экономическая мотивация должна быть вплетена в ткань процесса.

1. Являются ли стандарты организации достаточными для определения элементарных метрик таким образом, чтобы измерения, выполненные для разных проектов, могли сравниваться между собой?
2. Используются ли систематически результаты измерений, выполненных на предыдущих итерациях, при планировании последующих итераций?
3. Производятся ли SEPA текущие оценки зрелости процесса организации?
4. Производится ли измерение деятельности по усовершенствованию процесса в данной организации с точки зрения отдачи от инвестиций?
5. Действительно ли менеджеры проектов верят в то, что SEPA приносит реальную пользу их проекту?

E.5 ОБЩАЯ ОЦЕНКА ПРОЦЕССА

Ответы, приведенные в этой главе, позволяют предположить, что организация, которая реализовала схему процесса, описанную в данной книге, будет находиться на твердом уровне 3 SEI, а, возможно, и на уровне 4.

Верным признаком того, что организация достигла этого уровня зрелости, будет ее способность (1) предоставить развернутые ответы на вопросы документа с подробным объяснением специфичных для организации и проекта реализаций и (2) проиллюстрировать каждый ответ конкретными примерами из опыта в данной области. Если процедура подготовки к такой проверке процесса требует более одной или двух человеко-недель, то SEPA является в большей степени вывеской, чем фактором, приносящим пользу в работе организации. Дополнительное обеспечение, необходимое для проверки процесса, должно являться естественным побочным продуктом процесса управления проектом.

Глоссарий

Адаптируемость (adaptability) Тенденция к изменению объема доработок в зависимости от времени.

Архитектура (architecture) Существенная структура и поведение системы, включая все технические спецификации разработки, имеющие высокую степень точности.

База данных запросов на внесение изменений в ПО (software change order database) Хранимая совокупность описаний дискретных изменений, вносимых в базовую версию.

Бизнес-план (business case) Расходы, доходы, сроки и ожидаемая прибыль.

Бюджетные затраты (budgeted cost) Запланированный график расходов на весь жизненный цикл проекта.

Версия (release) Комплект рабочих продуктов, подлежащий оценке при достижении контрольной точки.

Внедрение, рабочий процесс (deployment workflow) Деятельность, связанная с передачей конечных продуктов пользователю.

Выполняемый код (executable code) Код на машинном языке, выполняемое ПО, а также сценарии сборки, сценарии инсталляции и данные, необходимые для использования продукта в конкретной среде выполнения.

Декомпозиция работ (work breakdown structure) Схема планирования; декомпозиция проекта на единицы работы, для которых могут быть определены или отслежены затраты, рабочие продукты и виды деятельности.

Демонстрация (demonstration) Набор программных компонентов, реализующий потоки событий в рамках демонстрируемых вариантов использования.

Дефекты (breakage) Усредненный показатель объема ПО, которое нуждается в доработке; в качестве единиц измерения используются строки исходного кода, функциональные точки, компоненты, подсистемы, файлы или другие единицы.

Динамика команды (team dynamics) Число уволенных и нанятых сотрудников в течение некоторого времени.

Доработка (rework) Средние затраты на внесение изменений, состоящие из трудозатрат на анализ, поиск решения и повторное тестирование всех изменений, вносимых в базовую версию ПО.

Жизненный цикл (life cycle) Один полный проход по четырем стадиям (начальная стадия, уточнение, конструирование и ввод в действие); охватывает время от старта начальной стадии до завершения стадии ввода в действие.

Завершенность (maturity) Зависимость MTBF от времени.

Заинтересованная сторона (stakeholder) Полномочный представитель, ответственный за принятие решений в каждой организации, имеющей свою долю от прибыли проекта.

Запрос на внесение изменений в ПО (software change order) Элементарная единица работ по внесению изменений в ПО, которая позволяет создавать, модифицировать или исключать компоненты в рамках базовой конфигурации.

Изменяющиеся уровни детализации (evolving levels of detail) Эволюция рабочих продуктов проекта, соответствующая текущему уровню понимания требований и архитектуры.

Интенсивность внесения изменений (change traffic) Количество запросов на внесение изменений, открытых и закрытых на протяжении всего жизненного цикла.

Исходный код (source code) Запись на языке программирования, представляющая собой реализацию компонентов и их форм, интерфейсов и зависимостей.

Итерационный процесс жизненного цикла (iterative life-cycle process) Процесс, последовательно уточняющий на протяжении нескольких итераций понимание проблемы, ее эффективное решение и план, гарантируя при этом баланс интересов всех заинтересованных сторон.

Итерация (iteration) Последовательность работ в рамках одной стадии, в результате выполнения которых создается версия; включает в себя подробный план и документированные результаты.

Кадровое обеспечение (staffing) Число работающих над проектом.

Компонент (component) Связный модуль ПО, который представлен в виде исходного кода или в выполняемом формате и для которого определены интерфейс и поведение.

Компонентно-ориентированная разработка (component-based development) Парадигма управления и разработки, отдающая предпочтение использованию существующих компонентов, а не разработке компонентов на заказ.

Комплект внедрения (deployment set) Рабочие продукты, представленные в виде кода на машинном языке и связанных с ним файлов.

Комплект проектирования (design set) Рабочие продукты, представленные в виде моделей области решений.

Комплект реализации (implementation set) Рабочие продукты, представленные в виде программ на языке программирования высокого уровня и соответствующих исходных файлов.

Комплект требований (requirements set) Рабочие продукты, представленные в виде организованного текста и модели проблемной области.

Комплект управления (management set) Рабочие продукты, содержащие планы проекта, описание промежуточных состояний и выполненной ранее работы.

Контрольная точка архитектуры жизненного цикла (life-cycle architecture milestone) Анализ, проводимый в конце стадии уточнения с целью демонстрации выполняемой архитектуры всем заинтересованным сторонам и достижения согласия по детальному плану стадии конструирования.

Контрольная точка версии продукта (product release milestone) Анализ, проводимый в конце стадии ввода в действие с целью оценки законченности ПО и, при необходимости, его передачи организации, осуществляющей сопровождение.

Контрольная точка начальной готовности к эксплуатации (initial operational capability milestone) Анализ, проводимый в конце стадии конструирования с целью оценки готовности ПО к вводу в действие у заказчика или пользователя и утверждения решения о начале квалификационного тестирования системы.

Контрольная точка целей жизненного цикла (life-cycle objectives milestone) Анализ, проводимый в конце начальной стадии с целью представления всем заинтересованным сторонам рекомендаций относительно проведения разработки. Включает в себя план, оценку стоимости и сроков, а также ожидаемую прибыль и экономию средств.

Конфигурация, базовая (configuration baseline) Поименованная совокупность программных компонентов и сопроводительной документации, которая является объектом управления изменениями и обновляется, сопровождается, тестируется, оценивается и устаревает как единое целое.

Конфигурируемый процесс (configurable process) Схема жизненного цикла, применимая для широкого спектра приложений.

Коэффициент дефектности (modularity) Зависимость среднего количества дефектов от времени.

Кривая затрат (expenditure profile) Зависимость затрат от времени.

«Круговая» разработка (round-trip engineering) Поддержка среды, необходимая для автоматизации и синхронизации рабочей информации в различных форматах (например, спецификации требований, проектные модели, исходный код, выполняемый код, варианты тестирования).

Модель требований (requirements model) Нотации для описания требований (например, UML) на различных уровнях абстракции для представления объектов и компонентов проблемной области, их атрибутов, статических связей, динамических взаимодействий и т.п.

Нотация, основанная на моделях (model-based notation) Семантически насыщенные графические и текстовые проектные нотации (например, UML).

Объективный контроль качества (objective quality control) Оценка процесса и всех промежуточных продуктов на протяжении жизненного цикла с использованием результатов четко определенных измерений, которые получаются непосредственно из рабочих продуктов и распространяются по всем видам деятельности и проектным командам.

Общая концепция (vision statement) Общее представление разрабатываемого продукта в виде, понятном всем заинтересованным сторонам.

Описание архитектуры ПО (software architecture description) Представления проектной модели, которые содержат информацию о структуре и поведении системы, достаточную для определения количества и спецификации ее составных частей, трудозатрат и других прямых затрат.

Описание версии (release description) Рабочий продукт, содержащий описание основных результатов версии.

Организационная политика (organizational policy) Рабочий продукт, определяющий элементы жизненного цикла и процесса: основные контрольные точки, промежуточные рабочие продукты, репозитории разработки, метрики, а также роли и распределение обязанностей.

Основная контрольная точка (major milestone) Событие масштаба системы, наступающее в конце каждой из стадий разработки и позволяющее ясно осознать проблемы системы в целом, синхронизировать процессы управления и разработки, а также подтвердить, что цели, стоявшие перед данной стадией, достигнуты.

Ответственный за проверку проекта (Project Review Authority, PRA) Лицо, персонально ответственное за то, чтобы проект по созданию ПО подчинялся всем организационным и экономическим правилам, практике и стандартам, касающимся ПО.

Ответственный за процесс разработки ПО (Software Engineering Process Authority, SEPA) Отдельное лицо или группа лиц, ответственная за сопровождение процесса, используемого в данной организации, и за облегчение процесса обмена информацией и руководствами между участниками проекта.

Ответственный за среду разработки ПО (Software Engineering Environment Authority, SEEA) Отдельное лицо или группа лиц, ответственная за автоматизацию процесса, сопровождение стандартной среды, обучение пользованию средой и сопровождение в организации наработок, пригодных для повторного использования

Оценка, рабочий процесс (assessment workflow) Вид деятельности, связанный с оценкой тенденций изменения качества процесса и продукта.

Оценка состояния (status assessment) Периодические мероприятия, обеспечивающие руководство частой и регулярной информацией относительно достигнутого прогресса.

План разработки ПО (software development plan) Вариант процесса, специфический для данного проекта.

Проверка (inspection) Анализ рабочих продуктов, выполняемый людьми.

Прогресс (progress) Объем работы, завершенной за некоторое время.

Продукт (product) Подмножество рабочих продуктов внедрения, передаваемое конечным пользователям.

Проектирование, рабочий процесс (design workflow) Деятельность, связанная с построением модели области решения, созданием архитектуры и рабочих продуктов проектирования.

Проектная модель (design model) Проектные нотации (например, UML) на различных уровнях абстракции для представления объектов и компонентов области решения, их атрибутов, статических связей, динамических взаимодействий и т.п.

Промежуточная контрольная точка (minor milestone) Событие уровня одной итерации, при котором выполняется подробный анализ содержания текущей итерации и утверждается решение о продолжении работ.

Прототип (prototype) Версия, которая необязательно является объектом управления изменениями и контроля конфигурации.

Работа (work) Усилия, которые необходимо приложить для выполнения определенного множества заданий.

Рабочий продукт (artifact) Дискретная связная совокупность информации, обычно разрабатываемая и рассматриваемая как единое целое.

Рабочий процесс (workflow) Поток тесно связанных и выполняемых последовательно видов деятельности.

Рабочие продукты внедрения (deployment artifacts) Проектные документы, необходимые для ввода программного продукта в эксплуатацию (например, руководство по эксплуатации компьютерной системы, руководство по установке ПО, планы и процедуры перехода в новую среду эксплуатации, инфраструктура вычислительной системы).

Реализация, рабочий процесс (implementation workflow) Деятельность, связанная с программированием компонентов и с развитием комплектов рабочих продуктов реализации и внедрения.

Риск (risk) Текущая или предварительно выявленная проблема, которая с высокой степенью вероятности может оказать неблагоприятное влияние на достижение основных контрольных точек.

Руководство пользователя (user manual) Справочная документация, необходимая для сопровождения поставляемого ПО.

Совет по контролю за конфигурацией (Configuration Control Board) Группа лиц, которая выступает в роли органа, принимающего решения по вопросам, касающимся содержимого базовой конфигурации.

Современный процесс (modern process) Процесс итерационной разработки ПО, в котором выполняется упреждающая разработка архитектуры, затем производится пошаговая реализация функциональных возможностей до получения окончательной версии продукта.

Спецификация версии (release specification) Рабочий продукт, содержащий область действия, план и цели версии.

Среда (environment) Автоматизация процесса, направленная на изготовление рабочих продуктов жизненного цикла. Должна включать в себя управление требованиями, визуальное моделирование, автоматизацию создания документации, средства программирования, автоматизированное регрессионное тестирование, а также управление изменениями и отслеживание дефектов.

Среда, рабочий процесс (environment workflow) Деятельность, связанная с автоматизацией производства рабочих продуктов жизненного цикла и с развитием среды сопровождения.

Среда для создания прототипов (prototyping environment) «Испытательный стенд» для создания прототипов архитектуры проекта с целью оценки соглашений, достигнутых на начальной стадии и стадии уточнения жизненного цикла.

Среда разработки (development environment) Полный набор инструментов разработки, необходимых для поддержки различных рабочих процессов и «круговой» разработки.

Среда сопровождения (maintenance environment) Развитая версия среды разработки.

Среднее время наработки на отказ (mean time between failures, MTBF) Среднее время работы системы между двумя сбоями типа 0 (критическими).

Стабильность (stability) Отношение числа открытых запросов на внесение изменений к числу закрытых запросов.

Стадия (этап) (stage) Часть жизненного цикла ПО, обладающая относительно однородной экономической моделью.

Стадия (фаза) (phase) Промежуток времени между двумя основными контрольными точками процесса, в течение которого достигается четко сформулированный набор целей, завершается создание определенных рабочих продуктов и принимается решение относительно перехода к следующей стадии.

Стадия ввода в действие (transition phase) Четвертая стадия жизненного цикла, в которой основное внимание уделяется процессу передачи продукта пользователям.

Стадия, начальная (inception phase) Первая стадия жизненного цикла, в которой основное внимание уделяется начальной концепции продукта и соответствующему бизнес-плану.

Стадия конструирования (construction phase) Третья стадия жизненного цикла, в которой основное внимание уделяется созданию полезного продукта, достаточно завершенного для ввода в действие у пользователей.

Стадия производства (production stage) Деятельность, которая осуществляется на последних этапах жизненного цикла и направлена на создание пригодных к использованию версий ПО в контексте базовых планов, требований и архитектуры, созданных на стадии разработки; в современном процессе должна быть связана с понятием экономии при больших масштабах.

Стадия разработки (engineering stage) Деятельность, которая осуществляется на ранних этапах жизненного цикла и объединяет планы, требования и архитектуру, разрешая, таким образом, риски, присущие разработке; обычно связана с понятием платы за масштаб.

Стадия сопровождения (maintenance stage) Развитие программного продукта после завершения жизненного цикла первоначальной разработки.

Стадия уточнения (elaboration phase) Вторая стадия жизненного цикла, в которой основное внимание уделяется созданию базовой архитектуры, согласованной с планом выпуска продукта и с требованиями.

Тип 0 запроса на внесение изменений (type 0 software change order) Критические сбои или фатальные проблемы ПО, непосредственно влияющие на возможность использования ПО по основному назначению.

Тип 1 запроса на внесение изменений (type 1 software change order) Ошибка или дефект, который либо не оказывает влияния на возможность использования ПО, либо может быть исправлен.

Тип 2 запроса на внесение изменений (type 2 software change order) Изменение, которое скорее может рассматриваться как улучшение, нежели попытка исправления дефекта.

Тип 3 запроса на внесение изменений (type 3 software change order) Изменение, вызванное сменой требований.

Традиционный процесс (conventional process) «Водопадный» процесс создания ПО, при котором происходит последовательный переход от анализа требований к проектированию, кодированию, тестированию модулей, интеграционному тестированию и верификации системы.

Требования, рабочий процесс (requirements workflow) Деятельность, связанная с анализом проблемной области и с созданием рабочих продуктов требований.

Управление, рабочий процесс (management workflow) Деятельность, связанная с планированием и контролем процессов жизненного цикла, а также с созданием условий для достижения успеха всеми заинтересованными сторонами.

Управление изменениями (change management) Отслеживание изменений в технических рабочих продуктах с целью контроля и понимания реального технического прогресса и тенденций изменения качества в процессе формирования приемлемого конечного продукта или промежуточной версии.

Упреждающая разработка архитектуры (architecture first) Подход, требующий достижения баланса между ведущими требованиями, архитектурно значимыми проектными решениями и планированием жизненного цикла прежде, чем будут выделены ресурсы на полномасштабную разработку.

Ссылки

[Boehm, 1981] Boehm, Barry W., *Software Engineering Economics* (Prentice-Hall, Englewood Cliffs, New Jersey, 1981). Используется с разрешения.

[Boehm, 1987] Boehm, Barry W., «Industrial Software Metrics Top 10 List», *IEEE Software*, Volume 4, Number 5 (September 1987), 84–85. Copyright ©1987 IEEE. Используется с разрешения.

[Boehm, 1988] Boehm, Barry W., «A Spiral Model of Software Development and Enhancement», *Computer*, Volume 21, Number 5 (May 1988), 61–72.

[Boehm, 1996] Boehm, Barry W., «Anchoring the Software Process», *IEEE Software*, Volume 13, Number 4 (July 1996), 73–82. Copyright ©1996 IEEE. Используется с разрешения.

[Boehm and Royce, Walker, 1988] Boehm, Barry W., and Walker E. Royce, «TRW IOC Ada COCOMO: Definition and Refinements», *Proceedings of the 4th COCOMO Users Group*, Pittsburgh, Pennsylvania (November 1988).

[Boehm *et al.*, 1995] Boehm, Barry W., Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby, «Cost Models for Future Software Engineering Processes: COCOMO 2", *Annuals of Software Engineering*, Volume 1 (1995), 57–94.

[Booch, 1996] Booch, Grady, *Object Solutions: Managing the Object-Oriented Project* (Addison-Wesley Publishing Company, Menlo Park, California, 1996).

[Brown, 1996] Brown, Norm, «Industrial-Strength Management Strategies», *IEEE Software*, Volume 13, Number 4 (July 1996), 94–103. Copyright ©1996 IEEE. Используется с разрешения.

[Davis, 1994] Davis, Alan M., «Fifteen Principles of Software Engineering», *IEEE Software*, Volume 11, Number 6 (November 1994), 94–96, 101. Copyright ©1994 IEEE. Используется с разрешения.

[Davis, 1995] Davis, Alan M., *201 Principles of Software Development* (McGraw Hill, New York, 1995). Copyright ©1995 McGraw Hill. Используется с разрешения.

[Defense Science Board, 1994] *Report of the Defense Science Board Task Force on Acquiring Defense Software Commercially* (The Undersecretary of Defense, Acquisition and Technology, Washington, D.C. June 1994).

[DeMarco, 1982] DeMarco, Tom, *Controlling Software Projects: Management, Measurement & Estimation* (Yourdon Press, Englewood Cliffs, New Jersey, 1982).

[DOD, 1988] DOD-STD-2167A, Defense System Software Development, Space and Naval Warfare Systems Command (December 1988). Отменен 5 декабря 1994 г.

[Horowitz, 1997] Horowitz, Ellis, *et al.*, «USC COCOMO II, 1997 Reference Manual», USC-CSE technical report (University of Southern California, Los Angeles, California, 1997).

[Humphrey, 1989] Humphrey, Watts S., *Managing the Software Process* (Addison-Wesley Publishing Company, Reading, Massachusetts, 1989).

[Humphrey, 1995] Humphrey, Watts S., *A Discipline for Software Engineering* (Addison-Wesley Publishing Company, Reading, Massachusetts, 1995).

[Jones, 1994] Jones, Capers, «The Economics of Object-Oriented Software», *American Programmer*, Volume 7, Number 10 (October 1994), 28–35.

[Jones, 1995] Jones, Capers, «Table of Programming Languages and Levels, Version 8", Software Productivity Research white paper (Burlington, Massachusetts), June 1995. Copyright ©1995 Capers Jones. Используется с разрешения.

[Jones, 1996] Jones, Capers, *Patterns of Software Systems Failure and Success* (International Thomson Computer Press, Boston, Massachusetts, 1996).

[Kruchten, 1995] Kruchten, Phillipe B., «The 4+1 View Model of Architecture», *IEEE Software*, Volume 12, Number 6 (November 1995), 42–50. Copyright ©1995 IEEE Software. Используется с разрешения.

[Royce, Walker, 1989] Royce, Walker E., «Ada Process Model Guidebook», Systems Engineering and Development Division Document TRW-TS-89-08 (TRW, Redondo Beach, California), November 1989. Используется с разрешения.

[Royce, Walker, 1990] Royce, Walker E., «Pragmatic Quality Metrics for Evolutionary Software Development Models», *Proceedings TRI-Ada '90*, Baltimore, Maryland (December 1990), 551–565.

[Royce, Walker, 1990b] Royce, Walker E., «TRW's Ada Process Model for Incremental Development of Large Software Systems», *Proceedings of the IEEE 12th International Conference on Software Engineering*, Nice, France, March 26–30, 1990.

[Royce, Walker, 1997] Royce, Walker E., «Managing Successful Iterative Development Projects: A Seminar in Software Best Practices», Version 2.3 (Rational Sofware Corporation, Menlo Park, California), 1997.

[Royce, Winston, 1970] Royce, Winston W., «Managing the Development of Large Software Systems», *Proceedings of IEEE WESCON* (August 1970), 1–9.

[SEI, 1993] *Capability Maturity Model for Software*, Version 1.1, Document No. CMU/SEI-93-TR-24, ESC-TR-93-177 (Carnegie Mellon University Software Engineering Institute, Pittsburgh, Pennsylvania, 1993). Используется с разрешения.

[SEI, 1993b] *Key Practices of the Capability Maturity Model*, Version 1.1, Document No. CMU/SEI-93-TR-25, ESC-TR-93-178 (Carnegie Mellon University Software Engineering Institute, Pittsburgh, Pennsylvania, 1993). Используется с разрешения.

[SEI, 1995] Carnegie Mellon University Software Engineering Institute, *The Capability Maturity Model: Guidelines for Improving the Software Process* (Addison-Wesley Publishing Company, Reading, Massachusetts, 1995).

[SEI, 1998] *Maturity Questionnaire*, Document No. CMU/SEI-94-SR-007 (Carnegie Mellon University Software Engineering Institute, Pittsburgh, Pennsylvania, June 1994). Используется с разрешения.

[Standish Group, 1995] The Standish Group,"Chaos", 1995. Используется с разрешения.