Глава 22

Управление масштабом и модели процесса разработки программного обеспечения

Основные положения

- Процесс разработки определяет *кто*, *что*, *когда* и *как* делает.
- В модели водопада деятельность по разработке программного обеспечения осуществляется с помощью последовательности шагов, причем каждый шаг основывается на результатах деятельности на предыдущем шаге.
- Спиральная модель начинается с создания набора основанных на рисках прототипов, после чего выполняется структурированный процесс, аналогичный модели водопада.
- Итеративный подход сочетает в себе свойства модели водопада и спиральной модели, а фазы жизненного цикла в нем отделяются от производимых в пределах каждой фазы действий по разработке.
- Независимо от используемой модели, необходимо разработать, как минимум, один ранний прототип, чтобы выяснить реакцию клиента.

До сих пор мы не обсуждали подробно процесс программной разработки в целом; в частности, не рассматривали, как сам процесс разработки влияет на способность команды достичь желаемых результатов. Однако понятно, что эффективное управление требованиями не может существовать без хорошо организованного процесса разработки, который полностью определяет множество действий, выполняемых командой при работе над программным продуктом. Некоторые процессы разработки программного обеспечения достаточно формальны, другие – неформальны, но процесс существует всегда, даже если он строго не определен и не документирован.

Процесс разработки программного обеспечения *определяет, кто (какой член команды), что (какие действия), когда (данные действия по отношению к другим действиям) и как (детали и этапы этих действий) делает для достижения цели.* Процессы разработки программного обеспечения оказывают заметное воздействие на способность команды разработать программу в срок и в пределах бюджета. В данной главе мы рассмотрим некоторые высокоуровневые аспекты различных процессов разработки программного обеспечения, в том числе временные фазы и основные типы деятельности во время этих фаз, а затем проанализируем, как сказывается на задаче управления масштабом проекта то, какой модели процесса следует команда.

"Модель водопада"

Бом (Boehm) (1988, a) отмечал, что еще в начале 1950-х, когда в программной отрасли осознали стоимость обнаружения программных дефектов на поздних этапах жизненного цикла, была принята логическая пошаговая модель процесса — от фазы разработки требований к фазе проектирования, кодирования и т.д. Это было значительным усовершенствованием по сравнению с существовавшей ранее двухфазной моделью "кодирования и исправления", согласно которой программисты сначала писали код, а затем поправляли его до тех пор, пока уже нечего было поправлять.

В 1970-х Уинстон Ройс (Winston Royce), работавший в компании TRW, предложил модель разработки программного обеспечения, известную как "модель водопада". В ней содержались следующие усовершенствования строго пошаговой модели.

- Появились петли обратной связи между стадиями; это отражает тот факт, что проектирование воздействует на разработку требований, а написание кода системы будет вызывать повторные обращения к проектированию и т. д.
- Параллельно с анализом требований и проектированием предлагалось разработать систему-прототип

Как показано на рис. 22.1, в модели водопада разработка программного обеспечения осуществляется посредством последовательности шагов. Каждый шаг основывается на действиях предыдущего шага. Проектирование логически следует за разработкой требований, кодирование — за проектированием и т.д. Модель водопада широко использовалась в последующие два десятилетия и успешно служила в качестве модели процесса для различных средних и больших проектов.

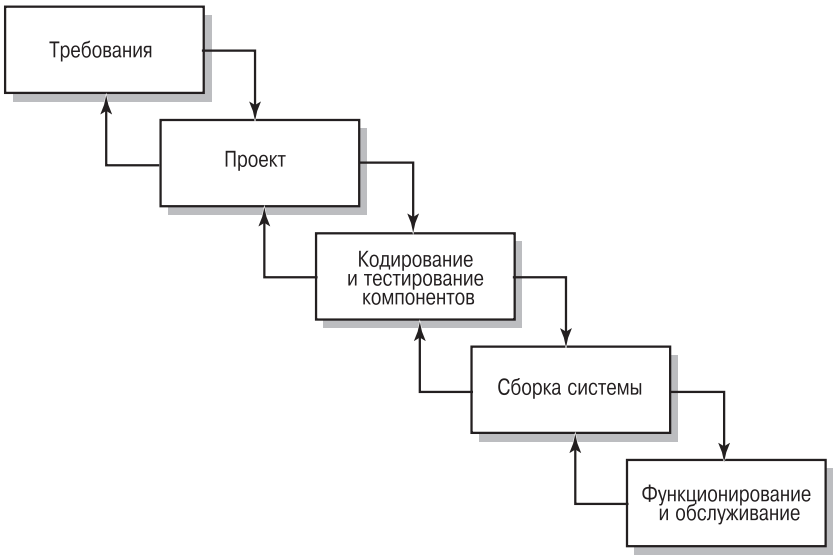

Рис. 22.1. "Модель водопада" процесса разработки программного обеспечения

Заметим, что в рис. 22.1 не указывается на необходимость создания прототипа (как, к сожалению, и было принято при применении данной модели). Это прискорбная ошибка, к которой мы еще вернемся.

В модели водопада возросла роль требований. Их разработка стала необходимым первым шагом при создании программного обеспечения, а также являлась основой проектирования и написания программного кода. Однако это же стало источником трудностей, так как в данной модели полная тщательная разработка требований и документов проектирования была обязательным условием окончания каждой из этих фаз. Кроме того, возможно, из-за неправильного применения слишком рьяными командами разработчиков, *эта модель стала олицетворять застывший косный подход к разработке, когда требования "заморожены" на время жизни проекта, изменения запрещены, а процесс разработки "живет" своей собственной жизнью.* В таком случае со временем команда может оказаться совершенно оторванной от реальности, на которой изначально основывался проект.

Дополнительные проблемы возникают, когда приходится решать задачу управления масштабом (рис. 22.2). Если же пытаться применять модель водопада к проекту, который изначально имеет масштаб 200%, результаты могут быть катастрофическими. К сроку сдачи ничто не работает, тестирование компонентов и сборка системы искусственно ускорены или не выполнены вообще, значительные средства затрачены на спецификацию, проектирование и написание кода тех функций системы, которые никогда не будут предоставлены. В результате нет ничего, что можно представить клиенту.

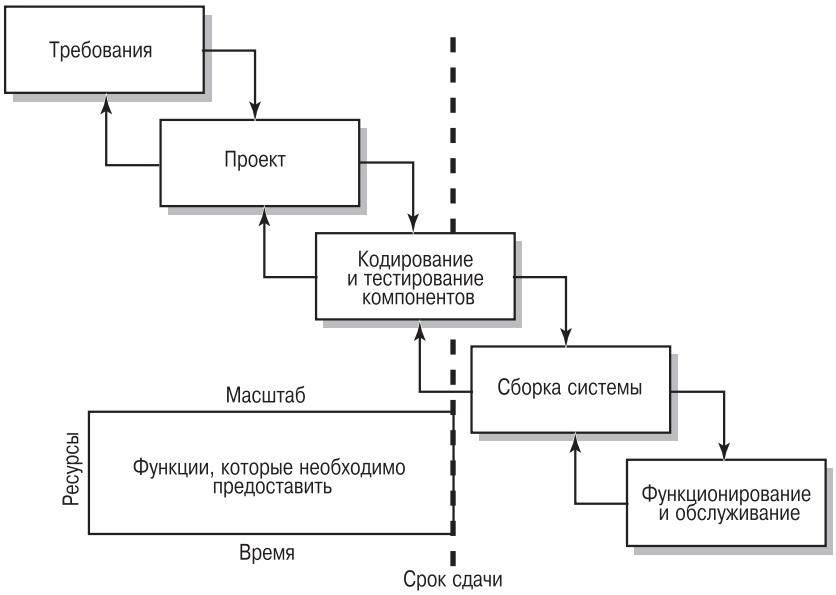

Рис. 22.2. Применение модели водопада к проекту с 200%-ным масштабом

В основном, именно эти причины привели к тому, что со временем модель водопада стала менее популярной. В результате вновь возникла тенденция переходить непосредственно к кодированию, не имея адекватного представления о требованиях к системе, что и было одной из основных проблем, которую пыталась решить модель водопада!

Спиральная модель

В своей работе Барри Бом (Barry Boehm) (1988, a) рекомендует другую схему процесса разработки программного обеспечения. Его "спиральной моделью" программной разработки руководствуются те, кто верит, что путь к успеху состоит из инкрементных разработок на основе рисков (рис22.3).

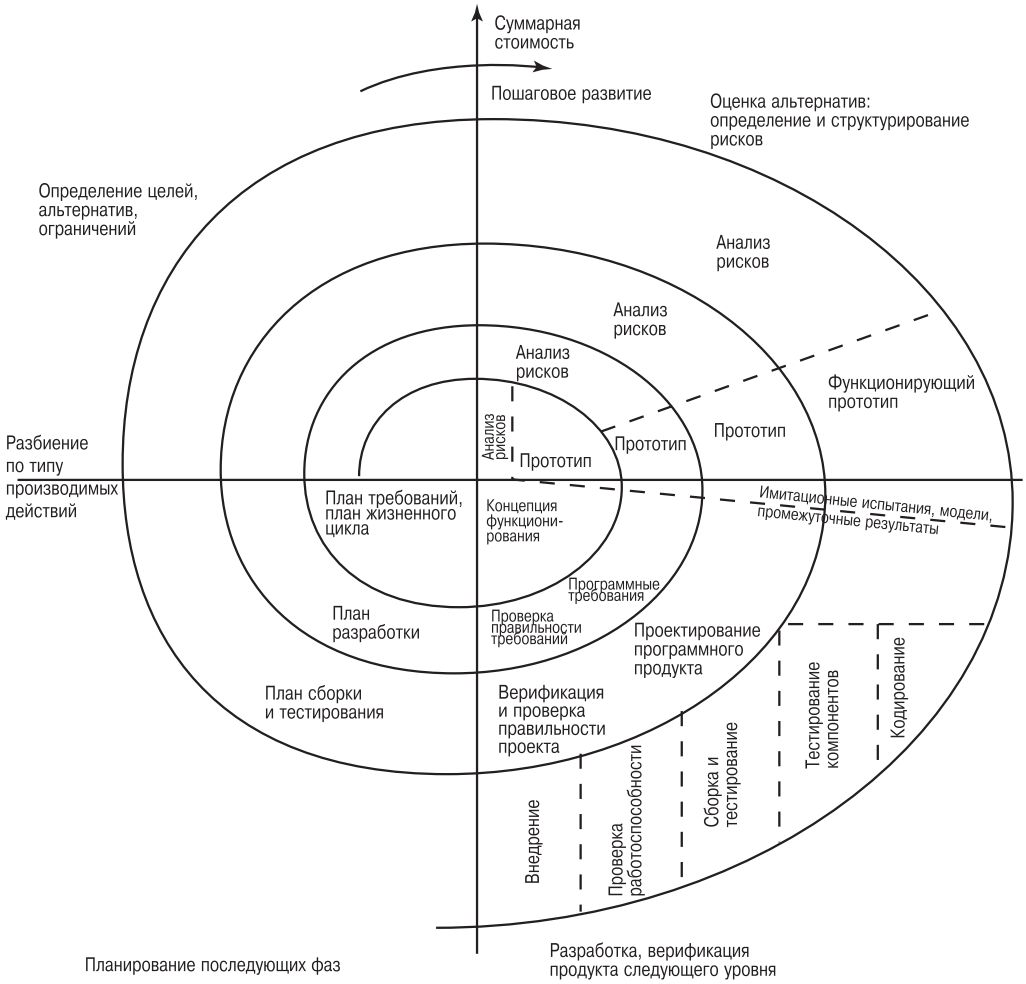

Рис. 22.3. Спиральная модель разработки

При использовании спиральной модели сначала создаются серии основанных на рисках прототипов, а затем проводится структурированный процесс построения конечной системы (аналогичный модели водопада). Конечно, при неправильном использовании спиральной модели могут возникать те же проблемы, что и в модели водопада. Проекты иногда сводятся к методу проб и ошибок; производятся отдельные составные части, которые необходимо наращивать и скреплять с помощью жевательной резинки или прово-

локи, что иногда еще называют процессом создания постоянно переделываемого кода. Преимущество заключается только в способности создавать необслуживаемый и непонимаемый код в два-три раза быстрее, чем при использовании более ранней технологии.

Однако если посмотреть на спиральную модель более внимательно, она может помочь в решении некоторых задач управления требованиями, рассматриваемых в данной книге. В частности, спиральная модель начинается с планирования требований и проверки правильности концепции, затем следует создание одного или нескольких прототипов, призванных на ранних этапах подтвердить наше понимание требований к системе. Положительной особенностью этого процесса является множество возможностей получения реакции пользователей и клиентов, что направлено на более раннее выявление синдрома "да, но…". Оппоненты этого строгого подхода отмечают, что в современных условиях непозволительной роскошью является тратить время на полную проверку концепции, два-три прототипа, а также на строгое следование модели водопада.

Рассмотрим, что произойдет, если спиральную модель применить к проекту с 200%-ным масштабом. Результат представлен на рис. 22.4. Приходится согласиться, что он немного лучше, чем при использовании модели водопада; с другой стороны, можно отметить, что к моменту сдачи, по крайней мере, один или два прототипа работоспособны и получены отзывы заказчика. (Конечно, значительная часть этих отзывов будет касаться отсутствия готового к тиражированию программного обеспечения!)

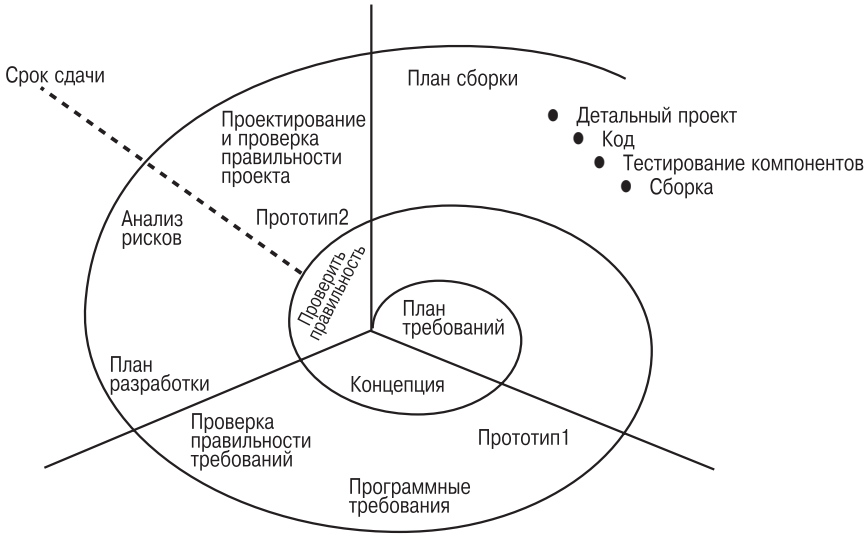

Рис. 22.4. Применение спиральной модели к проекту с 200%-ным масштабом

Итеративный подход

В 1990-е годы многие команды перешли к использованию нового подхода, который сочетает лучшие качества модели водопада и спиральной модели. Кроме того, он также содержит некоторые дополнительные конструкции из новой дисциплины инженерии программных процессов. Впервые предложенный Крачтеном (Kruchten, 1995), "итеративный подход" к настоящему времени хорошо описан в ряде книг, в том числе в

работах Крачтена (Kruchten, 1999) и Ройса (Royce, 1998). Этот подход доказал свою эффективность для широкого спектра проектов и имеет ряд преимуществ по сравнению с водопадной и спиральной моделями разработки.

В традиционных моделях процесса разработки развитие проекта осуществляется путем последовательного выполнения определенных действий, когда разработка требований предшествует проектированию, проектирование — реализации и т.д. Это достаточно разумно. В итеративном процессе фазы жизненного цикла отделяются от логической деятельности по разработке программного обеспечения, проводимой в каждой фазе, что позволяет повторно возвращаться к деятельности по разработке требований, проектированию и реализации на различных итерациях проекта. Кроме того, как и в спиральной модели, каждая итерация проектируется так, чтобы уменьшить всевозможные риски на данной стадии разработки.

Фазы жизненного цикла

Итеративный подход состоит из четырех фаз жизненного цикла: *начало, исследование, построение и внедрение*, что соответствует естественным состояниям проекта в указанные периоды времени (рис22.5).

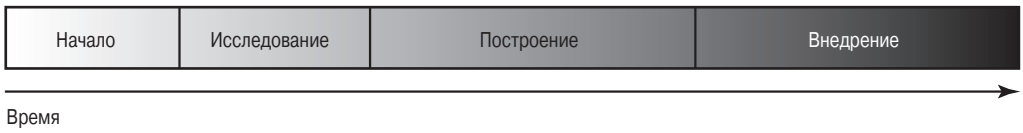

Рис. 22.5. Фазы жизненного цикла в итеративном подходе

1. *Начало.* На данной стадии команда уделяет основное внимание пониманию бизнес-варианта проекта, определению его масштаба, достижимости реализации. Производится анализ проблемы, создается документ-концепция (Vision document), предварительно оцениваются график, бюджет, а также факторы риска проекта.
2. *Исследование.* Уточняются требования к системе, задается исходная выполняемая архитектура и, как правило, разрабатывается и демонстрируется ранний прототип достижимости.
3. *Построение.* Главное внимание уделяется реализации. На этой стадии пишется бо́льшая часть программного кода, завершается проектирование и доработка архитектуры.
4. *Внедрение.* Производится бета-тестирование; пользователи и команда сопровождения системы получают опыт работы с приложением. Протестированная базовая версия приложения передается сообществу пользователей и разворачивается для использования.

Итерации

В каждой фазе проект, как правило, проходит множество итераций (рис. 22.6). *Итерация* — это приводящая к появлению некой выполняемой программы последовательность действий, для которой заданы план и критерий оценки. Каждая итерация основы-

вается на функциональных возможностях предыдущей итерации; таким образом, проект разрабатывается "итеративным инкрементным" методом.

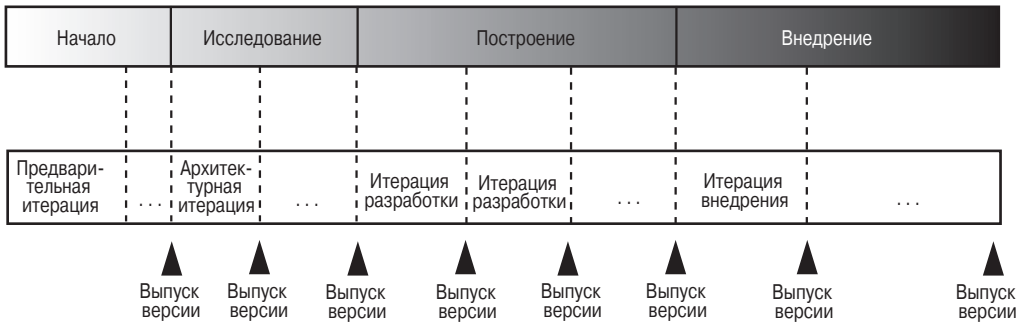

Рис. 22.6. Фазы и итерации, приводящие к появлению жизнеспособных версий

Существует ряд критериев отбора итераций. Ранние итерации следует разрабатывать для оценки жизнеспособности выбранной архитектуры, для некоторых самых важных прецедентов и прецедентов с высоким риском.

Рабочие процессы

В итеративном подходе деятельности, связанные с разработкой программного обеспечения, организованы в *рабочие процессы*. Каждый рабочий процесс состоит из логически связанного множества деятельностей и определяет, в каком порядке их следует проводить, чтобы создать жизнеспособный рабочий продукт или "артефакт". Хотя количество (и качество) рабочих процессов может варьироваться в зависимости от конкретной компании и проекта, обычно имеется, как минимум, шесть рабочих процессов (рис. 22.7).

На каждой итерации команда уделяет каждому рабочему процессу столько времени, сколько считает нужным. Таким образом, итерация может рассматриваться как мини-водопад, где представлены действия по разработке требований, анализу и проектированию и т. д., но каждый мини-водопад "настраивается" на специфические потребности данной итерации. Размеры "холмов" на рис. 22.7 отражают относительные трудозатраты на соответствующие рабочие процессы. Например, в фазе исследования значительное время уделяется уточнению требований и определению архитектуры, которая будет поддерживать функциональные возможности. Деятельности могут осуществляться последовательно (истинный мини-водопад) или параллельно, в зависимости от нужд конкретного проекта.

С точки зрения управления требованиями, итеративный подход имеет два существенных преимущества.

1. *Раннее получение "да, но…"*. В результате каждой итерации получается выполняемая версия; так что уже на самых ранних этапах проекта клиенты имеют возможность видеть рабочий продукт. Конечно же, их реакцией будет "да, но…", но на этой ранней стадии были затрачены только минимальные средства. При каждой последующей итерации размеры этих "но…" будут уменьшаться, и вы и ваш клиент постепенно подойдете к удовлетворительной системе.

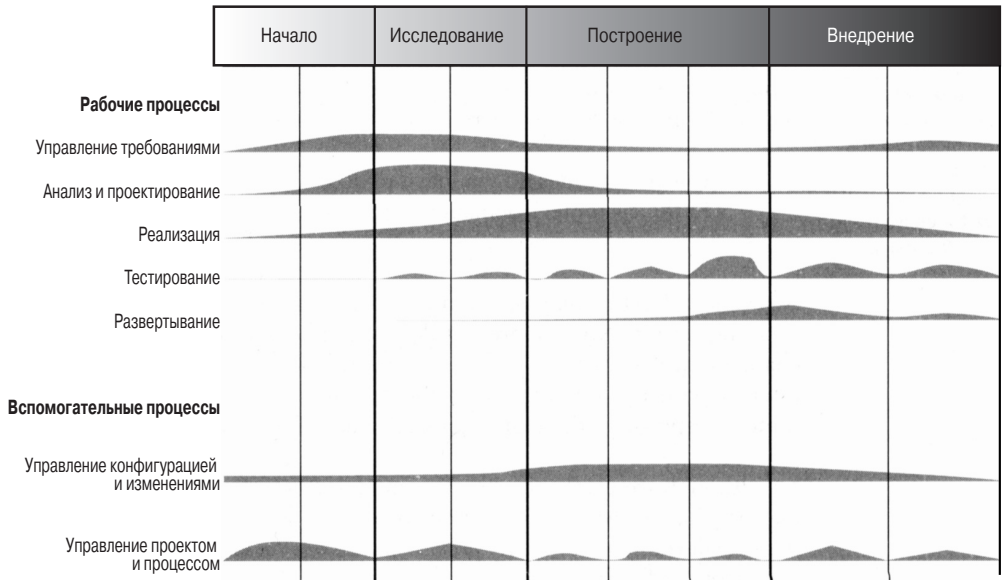

Рис. 22.7. Рабочие процессы итеративного подхода

2. *Легче управлять масштабом.* Если в первой итерации процент невыполнения составил 30% и более, это свидетельствует о том, что масштаб проекта, возможно, задан неправильно и его необходимо сократить. Однако даже с неверно заданным масштабом к сроку сдачи будет разработано несколько выполняемых итераций, а последняя может даже быть развернута. Даже если некоторых функциональных возможностей не хватает, версия будет представлять определенную ценность для пользователя. Если функции были правильно выбраны и упорядочены, это позволит заказчику достичь своих целей, по крайней мере частично, в то время как команда продолжит работу над итерациями разработки. Если архитектура устойчива и отвечает основным техническим требованиям, у команды будет надежная платформа для предоставления дополнительных функциональных возможностей.

Что делать, что делать...

> Независимо от того, какая модель используется, команда должна предложить по крайней мере один устойчивый оценочный прототип для раннего получения реакции клиента.

Одна из предпосылок данной книги состоит в том, что раннее получение реакции "да, но..." является одной из самых главных задач в процессе разработки программного обеспечения.

- Сколько раз придется делать прототип?
- Изменяются ли требования заказчика для каждого нового прототипа?
- Обречены ли мы на неудачу, независимо от того, какому процессу следуем?

Ответы таковы. Да, клиенты будут требовать изменений при каждом представлении. Нет, мы не обречены. Причина в том, что проблемы, связанные с внесением изменений, которые возникают после того, как клиент имеет возможность увидеть предлагаемую реализацию и поработать с ней, невелики по сравнению с важностью отзыва клиента на первый осязаемый артефакт процесса.

Таким образом, хотя мы предпочитаем использовать в наших разработках итеративную модель, независимо от того, какую модель процесса разработки используете вы, необходимо *обеспечить создание по крайней мере одного устойчивого оценочного прототипа* для получения реакции клиента до того, как будет выполнен основной объем действий по проектированию и написанию программного кода. (Помните о действиях по созданию прототипа, которые Ройс (Royce, 1970) изначально предлагал для модели водопада?!) Благодаря уменьшению числа изменений до разумного уровня разработчику удается путем внесения инкрементных изменений (как правило, в интерфейсы пользователя, отчеты и другие выходные результаты) создать качественный устойчивый технический проект и реализовать его. Затем тщательно организованный процесс завершения проектирования, кодирования, тестирования компонентов и сборки системы обеспечит для продукта прочный фундамент и в значительной мере будет способствовать действиям по обеспечению качества и тестированию.

Заключение части 4

В части 4, "Управление масштабом", мы выяснили, что проблема масштаба проекта является весьма типичной. Проекты, как правило, инициируются с объемом функциональных возможностей, вдвое превышающим тот, который команда может реализовать, обеспечив приемлемое качество. Это не должно нас удивлять: заказчики хотят большего, маркетинг хочет большего и мы также желаем большего. Тем не менее, нам необходимо ограничить себя, чтобы иметь возможность предоставить в срок хотя-*нибудь*.

Мы рассмотрели различные методы задания очередности выполнения (приоритетов) и ввели понятие базового уровня (совместно согласованного представления о том, в чем будут состоять ключевые функции системы как рабочего продукта нашего проекта) — понятие, задающее точку отсчета и ориентир для принятия решений и их оценки. Если масштаб и сопутствующие ожидания превышают реальные, в любом случае придется сообщать некие неприятные новости. Мы остановились на философии привлечения заказчика к процессу принятия трудных решений. В конце концов, мы являемся только исполнителями, а принимать решения должны наши заказчики, ведь это их проект. Поэтому вопрос стоит так: что *обязательно должно* быть сделано в следующей версии при имеющихся ресурсах проекта?

Даже в этом случае нам придется вести переговоры; в некотором смысле вся жизнь и, конечно, весь бизнес состоят из переговоров, и мы не должны этому удивляться. Мы кратко перечислили некоторые приемы ведения переговоров и намекнули, что они могут понадобиться команде.

Нельзя ожидать, что данный процесс полностью решит проблему масштаба, точно так же как никакой другой процесс в отдельности не решит проблемы разработки приложений. Но указанные шаги окажут заметное воздействие на размеры проблемы, позволят разработчикам приложения сконцентрировать свои усилия на критически важных подмножествах функций и в несколько приемов предоставить высококачественные системы, удовлетворяющие или превосходящие ожидания пользователей. Привлечение заказчика к решению проблемы управления масштабом повышает обязательства сторон, способствует взаимопониманию и доверию между заказчиком и командой разработчиков приложения. Имея всеобъемлющее определение продукта (документ-концепцию) и сократив масштаб проекта до разумного уровня, мы можем *надеяться* на успех в следующих фазах проекта.