Разработка требований к программному обеспечению

Содержание

Глава 5 Определение образа и границ проекта

Когда моя коллега Карен ввела в компании, где она работала, практику экспертизы документации требований, она обратила внимание, что многие проблемы, которые выявили проверяющие, касались объема проекта. Часто у экспертов было разное представление о предполагаемых границах и целях проекта. В результате им трудно было договориться о том, какое функциональное требование относится к спецификации требований.

Как мы уже видели в главе 1. бизнес-требования составляют высший уровень абстракции в цепи требований: они определяют образ и границы системы ПО. Пользовательские и функциональные требования к ПО должны находиться в соответствии с контекстом и целями, устанавливаемыми бизнес-требованиями. Требования, не содействующие достижению бизнес-целей проекта, не стоит включать в проект.

Проект, в котором нет четко определенного и согласованного направления, можно смело назвать кандидатом на провал. Участники проекта могут, сами того не осознавая, решать прямо противоположные задачи, если у них разные бизнес-цели и приоритеты. Лица, заинтересованные в проекте, никогда не смогут договориться о составе требований, если они не выработали общего понимания бизнес-целей. Четкое представление образа и границ проекта особенно важно при разработке сложного, распределенного ПО, когда географическое разделение участников замедляет ежедневное взаимодействие, упрощающее коллективную работу.

Признаком того, что бизнес-требования недостаточно ясно определены, можно считать ситуацию, когда определенные функции сначала включают в продукт, затем удаляют из него, а позднее снова добавляют. Проблемы, касающиеся образа и границ, необходимо разрешать до спецификации детализированных функциональных требований. Положение о рамках и ограничениях проекта необычайно полезно при обсуждении предлагаемых функций и целевых выпусков. Кроме того, на него можно ссылаться при принятии решений о изменении и расширении требований. В некоторых компаниях основные положения об образе и границах помещают на схему, которую приносят на каждое проектное совещание, чтобы все смогли быстро оценить, не выходит ли предложенное изменение за рамки проекта.

Определение образа продукта вплоть до бизнес-требований

Образ продукта (product vision) выстраивает работу всех заинтересованных лиц в одном направлении. Он описывает, что продукт представляет собой сейчас и каким он станет впоследствии. ***Границы проекта*** (project scope) показывают, к какой области конечного долгосрочного образа продукта будет направлен текущий проект. В положении о границах определена черта между тем, что входит в проект и тем, что остается вовне. То есть указанные рамки также определяют ограничения проекта. Более детально эти сведения изложены в базовой версии требований, которую разрабатывает команда для данного проекта.

Говоря об образе, мы подразумеваем весь продукт. Он будет изменяться относительно медленно при определении со временем стратегии продукта или развитии бизнес-целей. Границы же относятся к определенному проекту или его итерации, в которых реализуется чуть больше возможностей продукта, как показан на рис. 5-1. Границы более динамичны, чем образ, так как менеджер проекта изменяет содержимое каждой версии в соответствии с графиком, бюджетом, ресурсами и ограничениями качества. Задача планирования заключается в управлении границами определенного проекта (разрабатываемого или расширяемого), как определенным подмножеством большого стратегического образа. Положение об объеме для каждого проекта или каждой итерации или расширения продукта, вероятнее всего будет включено в спецификацию требований к этому ПО, а не в отдельный документ об образе и границах. Для крупных новых проектов необходимо создавать и полный документ об образе и границах проекта, и спецификацию требований. О шаблоне спецификации требований рассказано в главе 10.

Например, федеральное правительственное агентство получило долгосрочный заказ, рассчитанный на пять лет, на разработку массивной информационной системы. Команда определила бизнес-цели и образ системы еще на начальной стадии и за несколько следующих лет никак не меняла их. Было запланировано 15 отдельных выпусков конечной системы. Каждый

выпуск создавался как отдельный проект с описанием его собственных границ. Последние должны были соответствовать общему образу продукта и перекликаться с положениями о границах других проектов — это позволяло гарантировать, что ничего не будет пропущено.

Нереальные требования

Менеджер компании, специализирующейся на разработке ПО, сильно переживающая из-за почти катастрофического расползания границ проекта, с сожалением сказала мне; «Мы создали слишком нереальные требования». Она имела в виду, что любая идея, возникающая у члена команды, включалась в документ. У команды был четкий образ проекта, однако они не управляли границами, планируя несколько версий и откладывая определенные функции до более поздней (возможно, неопределенно поздней) версии. Наконец, после четырех лет разработки, черезмерно перегруженный продукт был выпущен. Толковое управление рамками проекта и постепенность в разработке позволили бы им поставить качественный продукт намного раньше.

Рис. 5-1. Образ продукта содержит запланированные версии оговоренного объема

Конфликтующие бизнес-требования

Бизнес-требования, собранные из разных источников, могут конфликтовать. Представьте себе компьютер со встроенным ПО, подсоединенный к Интернету и предназначенный для покупателей магазина; (киоск). При его разработке следует удовлетворить следующие бизнес-цели:

- получение дохода путем сдачи в аренду или продажи киоска продавцу;
- продажу потребительских товаров покупателям; привлечение внимания покупателей к торговой марке;
- предоставление широкого ассортимента товаров.

Бизнес-интересы розничного продавца таковы:

- максимальное увеличение дохода с доступного пространства;
- привлечение большего количества покупателей в магазин;
- увеличение объема продаж и уровня прибыли, если киоск заменит операции, выполняемые вручную.

Разработчику иногда хочется сделать киоск высокотехнологичным и увлекательным для покупателей. Продавцу же нужна простая и полностью готовая к эксплуатации система, а для покупателя важны удобство и функциональные возможности. Напряженность между этими тремя сторонами с их различными целями, ограничениями и бюджетом может привести к

несогласованности бизнес-требований. Тот, кто финансирует проект, должен разрешить эти конфликты до того, как аналитик детализирует системные требования и требования к ПО киоска. Основное внимание следует уделять фундаментальным задачам, сулящим наибольшую коммерческую выгоду («увеличатся продажи, будут привлечены новые покупатели»). При этом можно легко отвлечься НЕ внешние характеристики продукта («инновационный пользовательский интерфейс, привлекающий клиентов»), которые не отражают реальные бизнес-цели.

Кроме того, тот (или те), кто финансирует проект, должен разрешать конфликты различных заинтересованных в проекте лиц — не стоит ждать, что разработчики разберутся сами. По мере того как определяется все больше заинтересованных лиц и появляется все больше групп спонсоров с соперничающими интересами, увеличивается риск увеличения объема проекта. Неконтролируемый рост объема из-за того, что заинтересованные лица пытаются удовлетворить каждое пожелание, может привести к тому, что проект рухнет под собственным весом, так и не дав каких-либо ценных результатов. Разрешение подобных конфликтов часто удается лишь в результате политической и жестокой борьбы, что выходит за рамки тем, обсуждаемых в этой книге.

Бизнес-требования и варианты использования

Бизнес-требования определяют и набор бизнес-задач (вариантов использования), которые позволяет выполнять приложение *(ширина приложения),* и *глубину уровня,* до которого реализуется каждый вариант использования. Если бизнес-требования помогают вам определить, что некий вариант использования выходит за границы проекта, это значит, что вы принимаете решение о *ширине* проекта. *Глубина* простирается от простой реализации до полной автоматизации с множеством вспомогательных средств. Бизнес-требования позволяют понять, для каких вариантов использования необходима надежная и полная функциональность, а для каких достаточно поверхностной реализации, по крайней мере на первое время.

Бизнес-требования влияют на приоритеты реализации вариантов использования и связанные с ними функциональные требования. Например, такая бизнес-цель, как получение максимальной дохода от киоска, подразумевает реализацию на ранней стадии функций, отвечающих за продажу большего количества продуктов или предоставляющих услуги покупателям. Экзотичные, эффектные функции, привлекающие лишь немногих, жадных до технологичных новинок клиентов и не способствующие выполнению основной бизнес-задачи, не должны получать высокий приоритет.

Бизнес-требования также существенно влияют на способ реализации требований. Например, одна из причин создания Chemical Tracking System — уменьшить закупку новых бутылей с химикатами, используя те химикаты, которые уже есть на складе или в другой лаборатории. Опросы и наблюдения должны помочь выяснить, почему в данный момент это не делается. В свою очередь, на основе этой информации создаются функциональные требования и элементы дизайна, которые облегчают отслеживание химикатов в каждой лаборатории и помогают сотруднику, запрашивающему химикат, отыскивать искомое как можно ближе.

Документ об образе и границах проекта

Документ об образе и границах (vision and scope document) собирает бизнес-требования в единый документ, который подготавливает основу для последующей разработки продукта. В некоторых организациях с этой же целью создают устав проекта или положение о бизнес-задачах. Очень полезен и ***документ основных рыночных требований*** (market requirements document, MRD). В нем более детально, чем в документе об образе и границах, рассматриваются целевые сегменты рынка и проблемы, касающиеся коммерческого успеха продукта.

Владельцем документа об образе и границах считается тот, кто финансирует проект или несет аналогичную ответственность. Аналитик требований может вместе с этим человеком разрабатывать документ об образе и границах проекта. Информация, касающаяся бизнес-требований, должны поступать от лиц, четко понимающих, почему они взялись за проект. Это может быть клиент или топ-менеджер организации, разрабатывающей продукт, тот, кого привлекают технические новинки, менеджер по продукту, эксперт в данной предметной области

или специалисты отдела маркетинга.

На рис. 5-2 показан шаблон документа об образе и границах проекта. Шаблоны стандартизируют структуру документов, создаваемых проектными командами каждой организации. Как и в случае с любым шаблоном, измените его в соответствии со спецификой вашего проекта.

Может показаться, что части документа об образе и границах повторяются, однако они должны смыкаться. Рассмотрим пример.

1. Бизнес-требования
   1.1.Исходные данные
   1.2.Возможности бизнеса
   1.3.Бизнес-цели и критерии успеха
   1.4.Потребности клиента или рынка
   1.5.Бизнес-риски
2. Образ решения
   2.1.Положение об образе проекта
   2.2.Основные функции
   2.3.Предположения и зависимости
3. Масштабы и ограничения проекта
   3.1.Объем первоначально запланированной версии
   3.2.Объем последующих версий
   3.3.Ограничения и исключения
4. Бизнес-контекст
   4.1.Профили заинтересованных лиц
   4.2.Приоритеты проекта
   4.3.Операционная среда

Рис. 5-2. Шаблон документа об образе и границах проекта

Бизнес-шансы. Использовать слабо защищенные документы о конкурирующем продукте.

Бизнес-цель. Получить признание в качестве наиболее защищенного продукта на рынке, используя обзоры в отраслевых журналах и опросы потребителей, и захватить 80% рынка.

Пожелания потребителей. Более защищенный продукт.

Функция. Новый надежный механизм защиты.

1. Бизнес-требования

Проекты выпускаются с полным убеждением, что новый продукт для кого-то сделает мир лучше. Бизнес-требования описывают основные преимущества, которые новая система даст ее заказчикам, покупателям и пользователям. Для различных типов продуктов — информационных систем, коммерческих пакетов ПО и систем контроля, работающих в режиме реального времени, — выделяются различные преимущества.

1.1 Исходные данные

Суммирует обоснование и содержание нового продукта. Здесь помещают общее описание предыстории или ситуации, в результате которых было принято решение о создании продукта.

1.2 Возможности бизнеса

Для коммерческого продукта описывают существующие рыночные возможности и рынок, на котором продукту придется конкурировать с другими продуктами. Для корпоративной информационной системы описывают бизнес-проблему, которая разрешается посредством этого продукта, или бизнес-процессы, для улучшения которых требуется продукт, а также среду, в которой система будет использоваться. Кроме того, приведите здесь сравнительную оценку существующих продуктов и возможных решений, указав, в чем заключается привлекательность продукта и его преимущества. Опишите проблемы, которые не удается разрешить без продукта. Покажите, насколько он соответствует тенденциям рынка, развитию технологий или корпоративной стратегии. Кратко опишите другие технологии, процессы или ресурсы,

необходимые для удовлетворения клиента.

1.3 Бизнес-цели и критерии успеха

Суммирует важные преимущества бизнеса, предоставляемые продуктом, в количественном и измеряемом виде. В табл. 5-1 приведены примеры и финансовых и нефинансовых целей (Wiegers, 2002c). Если подобная информация приведена где-то еще, например в документе о бизнес-задачах, сошлитесь на этот документ, а не копируйте информацию. Определите, как заинтересованные лица будут определять и измерять успех проекта (Wiegers, 2002c). Установите факторы, которые максимально влияют на успех проекта — и те, которые организация может контролировать, и те, которые находятся вне сферы ее влияния, Определите меру для оценки того, были ли достигнуты бизнес-цели.

Таблица 5-1. Примеры финансовых и нефинансовых бизнес-целей

Финансовые	Нефинансовые
• Освоить X% рынка за Y месяцев. • Увеличить сектор рынка в стране X на Y% за Z месяцев. • Достигнуть объема продаж X единиц или дохода, равного $Y, за Z месяцев. • Получить X% прибыли или дохода по инвестициям в течение Y месяцев. • Достигнуть положительного баланса по этому продукту в течение Y месяцев. • Сэкономить $X в год, которые в настоящий момент расходуются на обслуживание системы. • Уменьшить затраты на поддержку на X% за Z месяцев. • Получить не более X звонков в службу обслуживания по каждой единице товара и Y звонков по гарантии каждой единицы товара в течение Z месяцев после выпуска товара. • Увеличить валовую прибыль для существующего бизнеса с X до У%.	• Достигнуть показателя удовлетворения покупателей, равного по крайней мере X, в течение Y месяцев со времени выпуска товара. • Увеличить производительность обработки транзакций на X% и снизить уровень ошибок данных до величины не более Y%. • Достигнуть определенного времени для достижения доминирующего положения на рынке. • Разработать надежную платформу для семьи связанных продуктов. • Разработать специальную базовую технологическую основу для организации. • Получить X положительных отзывов в отраслевых журналов к определенной дате. • Добиться признания продукта лучшим по надежности в опубликованных обзорах продуктов к определенной дате. • Соответствовать определенным федеральным и государственным постановлениям. • Уменьшить время оборота до X часов на Y% звонков покупателей в службу поддержки.

1.4 Потребности клиентов или рынка

Опишите потребности типичных покупателей или целевых сегментов рынка, включая потребности, которые не удовлетворяют настоящие продукты или информационные системы. Представьте проблемы, с которыми в настоящее время сталкиваются клиенты и которые решит новая система, и предоставьте примеры того, как покупатели будут использовать этот продукт. Определите на высоком уровне все известные важные требования к интерфейсу или производительности, но не касайтесь деталей проектирования или реализации.

1.5 Бизнес-риски

Обобщает важнейшие бизнес-риски, связанные с разработкой — или не с разработкой — этого продукта. В категории рисков входят рыночная конкуренция, временные факторы, приемлемость для пользователей, проблемы, связанные с реализацией, и возможные негативные

факторы, влияющие на бизнес. Оцените возможные потери от каждого фактора риска, вероятность его возникновения и вашу способность контролировать его. Определите все возможные действия по смягчению ситуации. Если вы уже подготовили эту информацию для анализа бизнес-задач или похожего документа, ссылайтесь на этот источник, а не копируйте эту информацию здесь.

2. Образ решения

В этом разделе документа определяется стратегический образ системы, позволяющей выполнять бизнес-задачи. Этот образ обеспечивает основу для принятия решений в течение жизненного цикла продукта. В него не надо включать детали функциональных требований или информацию, связанную с планированием проекта.

2.1 Положение об образе проекта

Составьте сжатое положение об образе проекта, обобщающее долгосрочные цели и назначение нового продукта. В этом документе следует отразить сбалансированный образ, удовлетворяющий различные заинтересованные лица. Он может быть несколько идеалистичным, но должен быть основан на существующих или предполагаемых рыночных факторах, архитектуре предприятия, стратегическом направлении развития корпорации или ограничениях ресурсов. Далее показан шаблон, состоящий из ключевых слов, который прекрасно подходят для документа об образе продукта (Moore, 1991):

- для [целевая аудитория покупателей];
- который [положение о потребностях или возможностях];
- эта (этот) [имя продукта]
- является [категория продукта]; а который(ая) [ключевое преимущество, основная причина для покупки или использования];
- отличие от [основной конкурирующий продукт, текущая система или текущий бизнес-процесс];
- наш продукт [положение об основном отличии и преимуществе нового продукта].

Вот как выглядит положение об образе для Chemical Tracking System в Contoso Pharmaceuticals, о которой говорилось в главе 2; ключевые слова выделены полужирным:

> ***Для*** *ученых,* ***которым*** *нужно запрашивать контейнеры с химикатом,* ***данная*** *Chemical Tracking System* ***является*** *информационной системой,* ***которая*** *обеспечит единую точку доступа к складу химикатов и к поставщикам. Система будет знать местоположение каждого контейнера с химикатом в компании, количество химиката в контейнерах и полную историю перемещения и использования каждого контейнера. Эта система сэкономит компании 25% затрат на химикаты в первый год работы, позволив полностью использовать уже полученные химикаты, ликвидировать меньшее количество частично истраченных или просроченных химикатов и применять единую стандартную систему приобретения химикатов.* ***В отличие*** *от действующих сейчас механизмов заказов химикатов, которые выполняются вручную,* ***наш продукт*** *будет генерировать все отчеты, необходимые для соответствия федеральным и государственным постановлениям, в которых требуются сведения об использовании, хранении и ликвидации химикатов.*

2.2 Основные функции

Назовите или пронумеруйте каждую основную функцию нового продукта или возможность, предоставляемую пользователям, уникальным последовательным способом, подчеркивая те из них, которые отличают его от предыдущих или конкурирующих продуктов. Дав каждой функции уникальное имя (в отличие от маркера абзаца), вы сможете отследить каждую функцию до отдельных требований пользователей, функциональных требований и других элементов систем.

2.3 Предположения и зависимости

Задокументируйте все предположения, сделанные заинтересованными лицами, когда они обдумывали проект и создавали данный документ об образе и границах. Часто предположения одних лиц не разделяют другие стороны. Если вы запишите их и просмотрите позже, TCI получите возможность обговорить основные положения проекта. Так вы избежите путаницы и ухудшения ситуации в будущем. Например спонсор Chemical Tracking System предположил, что новая система

заменит существующую систему складского учета и будет взаимодействовать с системой закупок компании Contoso. Также задокументируйте важнейшие зависимости проекта от внешних факторов — изменение (индустриальных стандартов или правительственных положений, других проектов, поставщиков со стороны или партнеров по разработке.

3. Масштабы и ограничения проекта

Когда химик изобретает новую химическую реакцию, которая преобразует один тип химиката в другой, он пишет документ, в который входит раздел «Рамки и ограничения», где описывает, что получится и не получится в результате этой реакции. Точно так же для проекта по разработке ПО следует определить его рамки и ограничения. Вам необходимо указать, что может делать система, а что ***не*** *может.*

> **Дополнительная информация**
>
> В главе 18 описывается, как воспользоваться атрибутом требований для документирования того, какие требования были отклонены или отложены.

Границы проекта определяют концепцию и круг действия предложенного решения. В ограничениях указываются определенные возможности, которые не будут включены в продукт. Рамки и ограничения помогают установить реалистичные ожидания заинтересованных лиц. Иногда клиенты запрашивают функции, слишком дорогостоящие или выходящие за предполагаемые границы продукта. Требования, выходящие за границы продукта, следует отклонять, если только они не настолько ценны, чтобы специально под них расширить *проект,* естественно, соответствующим образом изменив в бюджет, график и кадровый состав. Документируйте отклоненные требования и причины отказа от них, поскольку они имеют свойство появляться снова.

3.1 Объем первоначальной версии

Обобщает основные запланированные функции, включенные в первоначальную версию продукта. Опишите характеристики качества, которые позволят продукту предоставлять предполагаемые выгоды различным классам пользователей. Если ваша задача — сосредоточиться на разработке и уложиться в график, вам следует избегать искушения включить в версию 1.0 каждую функцию, которая когда-нибудь в будущем может понадобиться какому-то потенциальному покупателю. Увеличение сроков и сдвиг графика— типичный исход такого коварного расползания объема. Сосредоточьтесь на наиболее ценных функциях, имеющих максимально приемлемую стоимость, годных для самой широкой целевой аудитории, которые удастся создать как можно раньше,

Команда с которой мой коллега Скотт делал последний проект, решила, что пользователи должны иметь возможность запускать собственную службу доставки вместе с первой версии ПО. Версия 1.0 не обязательно должна быть быстрой, красиво оформленной или легкой в использовании, но она должна быть надежной; это основа работы команды. Первая версия системы выполняет лишь базовые задачи. В будущие выпуски будут включены дополнительные функции, возможности и средства, обеспечивающие легкость и простоту использования.

3.2 Объем последующих версий

Если вы представляете поэтапную эволюцию продукта, укажите, какие функции будут отложены и желательные сроки последующих выпусков. В последующих версиях вы сможете реализовать дополнительные варианты использования и функции и расширить возможности первоначальных вариантов использования и функций (Nejmeh и Thomas, 2002). Вы также сможете повысить производительность, надежность и другие характеристики качества по мере совершенствования продукта. Чем дальше вы заглядываете, тем более расплывчатыми будут границы проекта. Вам наверняка придется передвинуть функциональность с одного запланированного выпуска до другого и, возможно, добавлять незапланированные функции. Короткие циклы выпусков часто удобны для сбора отзывов клиентов.

3.3 Ограничения и исключения

Определение границы между тем, что входит и выходит за границы проекта, — отличный способ управления расползанием объема и ожиданиями клиентов. Перечислите все возможности или характеристика, которые могут ожидать заинтересованные в проекте лица, но включение которых в продукт или в определенную версию не запланировано.

4. Бизнес-контекст

В этом разделе обобщаются некоторые бизнес-проблемы проекта, включая профили основных категорий заинтересованных лиц и приоритеты управления.

4.1 Профили заинтересованных лиц

Заинтересованными в проекте лицами (stakeholders) называются отдельные лица, группы или организации, которые активно вовлечены в проект, на которых влияет результат проекта и которые сами могу влиять на этот результат (Project Management Institute, 2000; Smith, 2000). Профили заинтересованных лиц описывают различные категории клиентов и других ключевых лиц, заинтересованных в этом проекте. Вам не нужно описывать каждую группу заинтересованных лиц, например юристов, которые проверят соответствие надлежащим законам. Сферой вашего интереса должны стать различные группы клиентов, целевые рыночные сегменты и различные классы пользователей, входящих в эти сегменты. В профиль каждого заинтересованного в проекте лица включается следующая информация:

- основная ценность или преимущество, которое продукт принесет заинтересованным лицам и то, как продукт удовлетворит покупателей. Ценность для заинтересованных лиц представляют:
  - улучшенная производительность;
  - меньшее количество переделок;
  - снижение себестоимости;
  - ускорение бизнес-процессов;
  - автоматизация задач, ранее выполнявшихся вручную;
  - возможность выполнять совершенно новые задачи;
  - соответствие соответствующим стандартам и правилам;
  - лучшая, по сравнению с текущими продуктами, легкость и простота использования;
- их вероятное отношение к продукту;
- наиболее интересные функции их характеристики;
- все известные ограничения, которые должны быть соблюдены.

4.2 Приоритеты проекта

Чтобы принимать эффективные решения, заинтересованные лица должны договориться о приоритетах проекта. Один из подходов к этому заключается в рассмотрении пяти измеряемых параметров проекта: функции (или объем), качество, график, затраты и кадры (Wiegers. 1996a). В любом проекте каждый из этих параметров относится к одной из трех категорий:

ограничение — лимитирующий фактор, в рамках которого должен оперировать менеджер проекта;

ключевой фактор — важный фактор успеха, ограниченно гибкий при изменениях;

степень свободы — фактор, который менеджер проекта может до определенной степени изменять и балансировать относительно других параметров.

Задача менеджера проекта — настроить те факторы, которые представляют собой степени свободы для достижения ключевых факторов успеха проекта в рамках, налагаемых ограничениями. Не все факторы могут быть ключевыми, как и не все — ограничениями. Менеджеру проекта необходима определенная степень свободы для того, чтобы он мог реагировать должным образом на изменение требований к проекту или внешних обстоятельств. Представьте себе, что отдел маркетинга неожиданно требует создать продукт на месяц раньше сроке. Какова будет ваша реакция?

Вы отложите реализацию определенных требований до более поздней версии?

Сократите запланированный цикл тестирования системы?

Оплатите сверхурочную работу вашим специалистам или пригласите специалистов по контракту для ускорения разработки?

Привлечете ресурсы других проектов для разрешения ситуации? Именно от приоритетов проекта зависят ваши действия в подобных ситуациях.

4.3 Операционная среда

Опишите среду, в которой будет использоваться система, и определите важнейшие требования к доступности, надежности, производительности и целостности. Эта информация существенно влияет на определение архитектуры системы, что является первым — и часто самым важным— этапом проектирования. Архитектура системы, предназначенной для поддержки пользователей, которые находятся далеко друг от друга и которым необходим круглосуточный доступ, сильно отличается от той, что предназначена для доступа пользователей, находящихся рядом, только в рабочие часы. На нефункциональные требования, такие как отказоустойчивость и способность обслуживать систему во время ее работы, требуется значительное количество средств, отпущенных на проектирование и реализацию. Чтоб прояснить ситуацию, задайте заинтересованным лицам уточняющие вопросы.

Пользователи расположены далеко (географически) или близко друг от друга? В скольких часовых поясах работают ваши пользователи?

Когда пользователям, находящимся в различных географических местоположениях, требуется доступ к системе? Где данные генерируются и используются? Насколько далеко друг от друга расположены эти местоположения? Нужно ли объединять данные из разных местоположений?

Известно ли максимальное время отклика для получения доступа к данным, которые могут храниться удаленно?

Готовы ли пользователи смириться с прерыванием работы службы или непрерывный доступ к системе крайне важен для работы их компании?

Какие элементы управления безопасностью и требования к защите данных необходимы?

Контекстная диаграмма

Уточнение рамок определяет границу и связи системы, которую мы разрабатываем, со всем остальным миром. ***Контекстная диаграмма*** (context diagram) графически иллюстрирует эту границу. Она определяет ***оконечные элементы*** (terminators), расположенные вне системы, которые определенным образом взаимодействуют с ней, а также данные, элементы управления и материальные потоки, протекающие между оконечными элементами и системой. Контекстная диаграмма представляет собой высший уровень абстракции в диаграмме потока данных, разработанной по принципам структурного анализа (Robertson и Robertson, 1994), но эта модель полезна и в случае применения какой-либо другой методики разработки. Вы можете включить контекстную диаграмму в документ об образе и границах, или определить ее как приложение к спецификации требований, или как часть модели потоков данных системы.

На рис. 5-3 показана часть контекстной диаграммы для Chemical Tracking System. Вся система изображена кружком; на контекстной диаграмме намеренно не показывают внутренние объекты системы, процессы и данные. «Система» внутри кружка может иметь любую комбинацию ПО, оборудования или людских ресурсов. Оконечные элементы в прямоугольниках представляют классы пользователей («Химик» или «Покупатель»), отделы («Отдел охраны труда и техники безопасности»), другие системы («База данных по обучению») или аппаратные устройства («Считывающее устройство штрих-кода»). Стрелками показаны потоки данных («запрос химиката») или физические элементы («контейнер с химикатом») между системой и оконечными элементами.

Вы можете ожидать, что поставщики химикатов должны быть показаны на диаграмме в виде оконечных элементов. Ведь компания направляет заказы для выполнения поставщикам, а те отправляют контейнеры с химикатами и счета в Contoso Pharmaceuticals, отдел же закупок пересылает чеки продавцам. Однако эти процессы происходят вне Chemical Tracking System, как часть операций отделов закупок и приобретений. Глядя на контекстную диаграмму становится

совершенно ясно, что система не участвует напрямую в размещении заказов у поставщиков, в получении продуктов или оплате счетов.

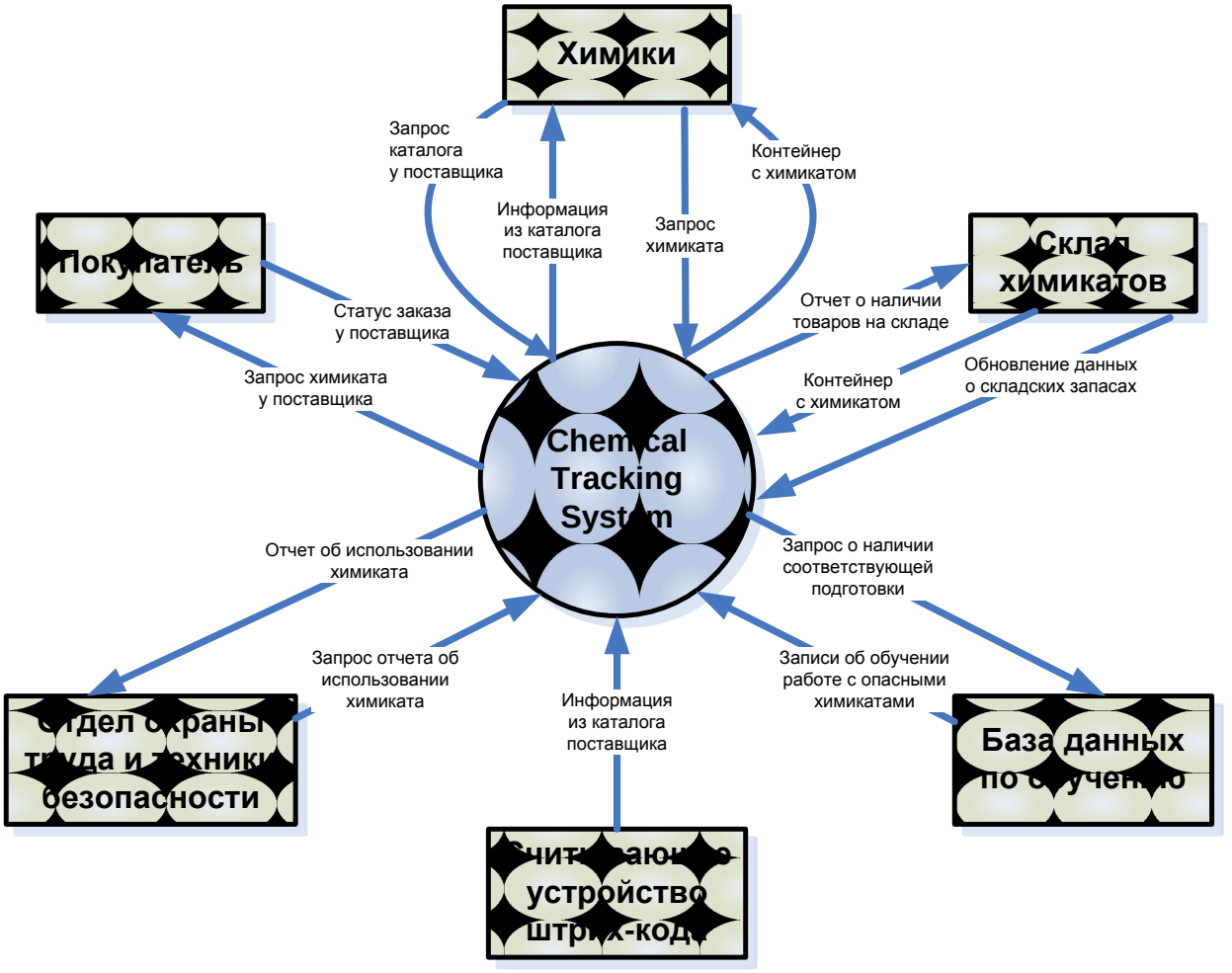


Рис. 5-3. Контекстная диаграмма для Chemical Tracking System

Назначение таких средств, как контекстная диаграмма, заключается в стимулировании ясного и точного взаимодействия между заинтересованными в проекте лицами. Эта ясность гораздо важнее слепого следования правилам создания «правильной» контекстной диаграммы. Однако я горячо рекомендую использование схему, показанную на рис. 5-3, в качестве стандарта, когда вы возьметесь рисовать контекстные диаграммы. Предположим, вы решите использовать треугольник вместо кружка для изображения системы и эллипсы вместо прямоугольников для изображения оконечных элементов. Вашим коллегам будет трудно читать диаграмму, нарисованную в соответствии с вашими личным предпочтениями, а не с общепринятым стандартом.

Не упускайте границы из вида

Бизнес-требования и понимание того, как клиенты будут использовать продукт, ценны при расползании границ. В изменении объема как такового нет ничего плохого, если это помогает вам направить проект в сторону удовлетворения развивающихся потребностей клиентов. Документ об образе и границах позволяет оценить, действительно ли предложенные функции и требования стоит включать в проект. Помните, каждый раз, когда кто-то выдвигает новое требование, аналитик должен спросить: «Попадает ли оно в рамки проекта?»

Один из возможных ответов — признать, что требование явно выходит за границы проекта. Оно может быть интересным, но его следует переадресовать будущим версиям или другому проекту. Другая возможность — требование попадает в границы проекта. Вы можете включить

такие требования в проект, если они обладают высоким приоритетом по сравнению с теми требованиями, которые уже описаны в проекте. При этом зачастую приходится решать, отложить или отклонить другие запланированные требования.

Третья возможность — предложенное новое требование выходит за границы объема, но идея так хороша, что следует изменить границы проекта, чтобы требование попало в него. То есть существует обратная связь между пользовательскими требованиями и бизнес-требованиями. При этом вам придется обновить документ об образе и границах проекта, изменения в котором должны контролироваться с момента создания его базовой версии. При увеличении объема проекта, как правило, приходится пересматривать запланированные бюджет, ресурсы, график и, возможно, кадры. В идеале первоначальный график и ресурсы можно приспособить к определенным изменениям благодаря хорошо продуманному резерву для непредвиденных обстоятельств (Wiegers, 2002d). Однако, если вы не учли в бюджете статью роста требований, вам придется перепланировать бюджет после утверждения изменений в требованиях.

Управление объемом и разработка по графику

Энрике, менеджер проекта в Lightspeed Financial Systems, занимался выпуском Интернет-версии ведущего программного продукта Lightspeed для управления ценными бумагами. Потребовались бы годы для полного размещения развитого приложения, однако компания должна была появиться в Интернете сейчас. Энрике выбрал прием «разработка по графику» и обещал выпускать новую версию каждые 90 дней (McConnelL, 1996). Его команда специалистов по маркетингу расставила приоритеты требований к продукту. В спецификации требований для каждой ежеквартальной версии входит утвержденный набор новых и улучшенных функций, а также список «растягиваемых» требований низкого приоритета, которые предполагалось реализовать, если позволяло время. Команда Энрике не включала каждое «растягиваемое» требование в каждую версию, но и они поставили новую, стабильную версию каждые три месяца с помощью подхода, основанного на графике и управлении объемом. График и качество — это единственные ограничения проекта «разработка по графику», причем объем представляет собой степень свободы.

Из-за расползания объема часто необходимо переделывать уже выполненную работу, чтобы учесть изменения. Если при добавлении новой функциональности не увеличить распределенные ресурсы или время на проект, то страдает качество. Задокументированные бизнес-требования упрощают управление разумным ростом объема, когда необходимы изменения на рынке или в бизнесе. Они также помогают расстроенному менеджеру проекта оправдаться, если ему приходится говорить «нет» — или по крайне мере «еще нет», — когда влиятельные лица пытаются включить все больше и больше функций в чрезвычайно ограниченный проект.

Что теперь?

- Попросите нескольких заинтересованных в проекте лиц написать положение об образе, используя шаблон ключевых слов, о котором говорилось в этой главе. Посмотрите, насколько совпадают их представления. Разрешите любые нестыковки и составьте объединенное положение об образе, согласовав его со всеми заинтересованными лицами.
- Независимо от того, скоро ли выпуск нового проекта или вы еще в середине процесса сборки, напишите документ об образе и границах, используя шаблон на рис. 5-2, и попросите остальных членов команды просмотреть его. Может выясниться, что у членов команды нет единого представления об образе или границах продукта. Решите эту проблему сейчас, не оставляйте ее в «свободном плавании»; в будущем исправить ее гораздо труднее. Таким образом, есть несколько способов изменения шаблона, что позволит наилучшим образом удовлетворить различные проекты, выполняемые вашей организацией.

Глава 6 Как отобрать пользователей для работы над проектом

Если вы разделяете мое убеждение, что критический взгляд клиента на разрабатываемое ПО крайне важен для создания продукта отличного качества, вы с самого начала должны привлечь клиентов к работе над проектом. Качество собранных требований к ПО, а следовательно, и успех самого ПО зависит от того, насколько хорошо голос клиента будет услышан разработчиками. Прежде всего необходимо понять, чего клиент хочет:

- определите различные классы пользователей вашего продукта;
- определите основные источники получения информации о потребностях клиентов;
- выберите представителей каждого класса пользователей и прочих групп заинтересованных лиц и поработайте с ними;
- согласуйте с ними, кто будет отвечать за принятие решений по проекту.

Вовлечение клиентов в работу над проектом — единственный способ избежать их претензий и разочарования, когда они получают не тот продукт, который ожидали. Недостаточно просто расспросить нескольких клиентов о том, как они представляют продукт, чтобы немедля браться за дело. Если разработчики будут выполнять все требования клиентов, им скорее всего только и придется, что переделывать продукт, поскольку клиенты зачастую сами не знают, чего им требуется на самом деле.

Желания пользователей не всегда совпадают с функциональностью, необходимой для выполнения задач с помощью конкретного продукта. Чтобы получить более точное представление о потребностях пользователей, аналитик требований должен опросить клиентов, проанализировать информацию, внести уточнения и определить, что именно нужно пользователям для выполнения работы. На аналитика возложена вся полнота ответственности за документирование функций и свойств системы и за ознакомление с этой информацией всех заинтересованных лиц. Это постепенный процесс, требующий времени. Если вы не желаете тратить его на выработку согласованного представления о желаемом продукте, готовьтесь к неизбежным переделкам, срывам графика и недовольству клиентов.

Основные источники получения информации о потребностях клиентов

Способы и источники получения информации от клиентов зависят от специфики продукта и среды разработки. Необходимо выслушивать разные точки зрения и выбирать различные

источники информации, а это лишний раз подтверждает, что работа по сбору требований тесно связана с общением. Вот несколько типичных источников получения информации при сборе требований к ПО.

Опросы потенциальных пользователей и дискуссии с ними. Самый очевидный способ выяснить потребности потенциальных пользователей нового программного продукта — опросить их. В этой главе рассказывается, как выбрать толковых представителей пользователей, а в главе 7 — как выяснить, что же им действительно необходимо.

Документы, где описан уже работающий или конкурирующий продукт. Эти документы могут также содержать корпоративные или отраслевые стандарты, которых необходимо придерживаться, а также постановления и законы, которым должен соответствовать продукт, Пригодятся и описания уже реализованных и будущих бизнес-процессов. Опубликованные сравнительные обзоры позволяют выявить недостатки других аналогичных продуктов, на которые стоит вовремя обратить внимание, чтобы получить конкурентное преимущество,

Спецификации требований к системе. Для продукта, включающего программные и аппаратные компоненты, создается спецификация требований к системе, описывающая продукт в целом. Отдельные требования к системе в целом касаются каждой подсистемы (Nelsen, 1990). Аналитик может вычленить дополнительные, более мелкие функциональные требования из требований к конкретной подсистеме.

Отчеты об ошибках и претензии к возможностям работающей системы. Персонал внутрикорпоративной и выездной службы поддержки — ценный источник информации. Они в курсе проблем, с которыми сталкиваются пользователи работающей системы, и постоянно выслушивают идеи клиентов по совершенствованию ее следующей версии.

Маркетинговые исследования и опросы пользователей. Опросив массу потенциальных пользователей продукта, вы получите кучу информации. Проконсультируйтесь с экспертом по планированию и управлению опросами, дабы убедиться, что задаете нужные вопросы нужным людям (Fowler, 1995). Опрос позволяет проверить, насколько четко вы понимаете требования, которые собираете или, как вы думаете, уже выявили, однако это не лучший способ стимулировать творческое мышление. Прежде чем начать опрос, необходимо критически изучить предполагаемые вопросы. Велико же будет ваше разочарование, когда вы поздно обнаружите, что один из вопросов сформулирован неоднозначно или что важного вопроса в списке не оказалось.

Наблюдение за пользователями на рабочих местах. Наблюдая «один рабочий день из жизни пользователя», аналитик выявляет особенности работы действующей системы, а также потребности потенциальных пользователей будущей системы. Такое исследование позволяет проверить информацию, собранную в ходе интервью, определить темы новых опросов, выявить проблемы действующей системы и понять, какие функции новой системы необходимы для автоматизации операций (McGraw и Harbison, 1997; Beyer и Holtzblatt, 1998). Наблюдая за работой пользователей, вы лучше и полнее поймете суть рабочего процесса, чем если просто попросите их описать все этапы их работы. Излагая и обобщая данные, аналитик должен абстрагироваться от ситуации и рассматривать ее несколько «сверху; это гарантирует, что собранные требования будут представлять интересы класса пользователей в целом, а не отдельных лиц. Кстати, опытные аналитики зачастую способны предложить что-то весьма дельное для совершенствования текущих бизнес-процессов.

Сценарий анализа задач пользователей. Определив, какие задачи пользователю требуется выполнять средствами системы, аналитик должен выработать необходимые функциональные требования к системе. Это — суть подхода, основанного на применении вариантов использования, описанного в главе 8. Не забудьте указать, какие данные необходимы и генерируются в ходе выполнения задачи, а также источники этих данных.

События и реакция на них. Перечислите внешние события и соответствующую реакцию системы на них. Данный способ особенно хорош для систем реального времени, которые считывают и обрабатывают потоки данных, коды ошибок, управляющие сигналы и сигналы прерывания от внешних устройств.

Классы пользователей

Помимо всего прочего, пользователей продукта можно подразделять по таким признакам:

- по частоте использования продукта;
- по опыту в предметной области и опыту работы с компьютерными системами;
- по требуемой им функциональности;
- по задачам, которые им приходится выполнять;
- по правам доступа к системе (например, обычный пользователь, гость или администратор).

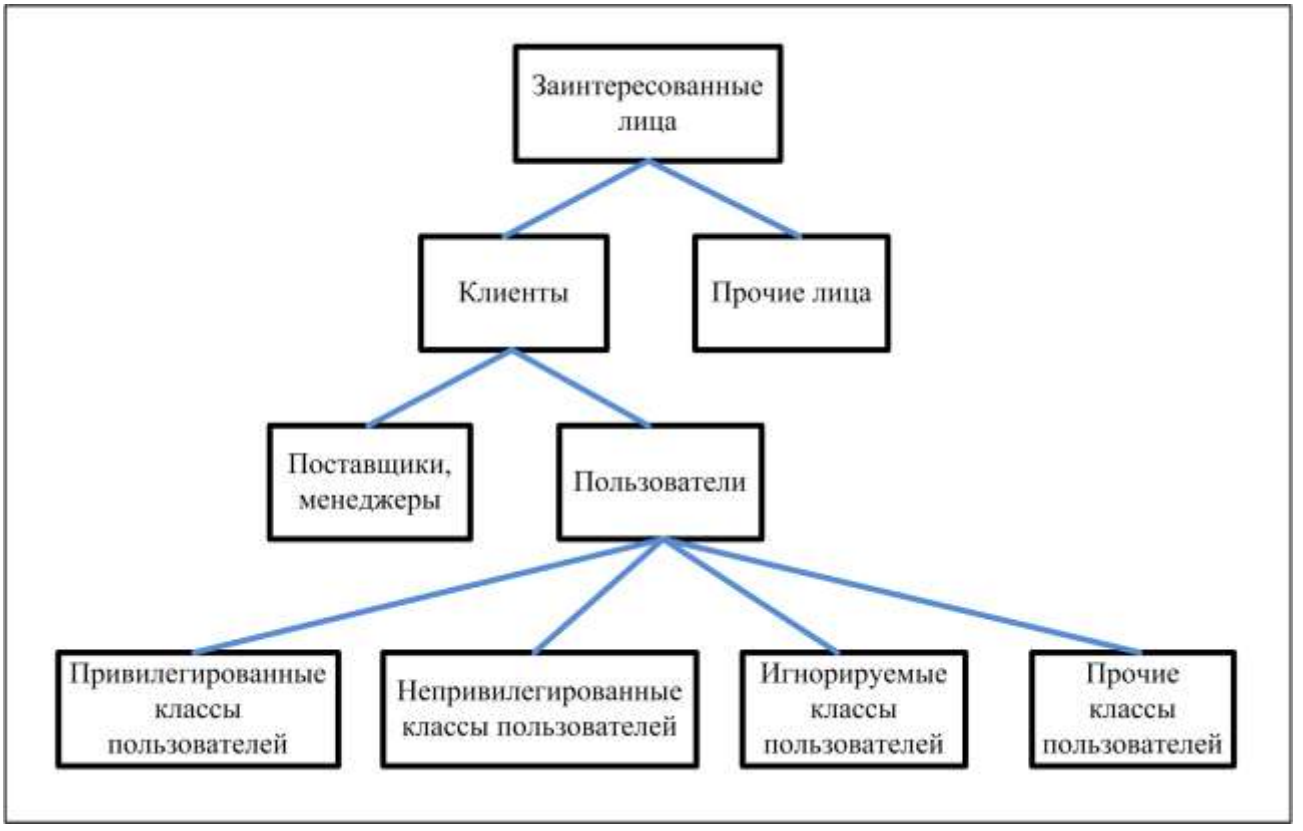


Рис. 6-1. Иерархия клиентов, пользователей и заинтересованных лиц

На их основе формируются классы пользователей. Некоторых сотрудников можно отнести к нескольким классам. Так, администратор иногда работает с системой, как рядовой пользователь. Конечные пользователи ПО вне системы, показанные на контекстной диаграмме в главе 5, представляют собой потенциальных кандидатов на отдельные классы пользователей. Это часть непосредственных пользователей продукта из большой группы клиентов, которых можно выделить из всех заинтересованных в проекте лиц (рис. 6-1).

Очень заманчиво поделить пользователей на классы по их географическому положению, виду бизнеса, которым занимается компания, или занимаемой должности, а не на основании того, как они взаимодействуют с системой. Например, компания — производитель банковского ПО первоначально предполагала разделить пользователей по типам финансовых учреждений, в которых те работают: крупный коммерческий банк, небольшой коммерческий банк, ссудо-сберегательная общество или кредитный союз. В действительности же таким образом выделяются потенциальные сегменты рынка, а не различные классы пользователей. Во всех этих финансовых учреждениях у людей, занимающихся ссудами, формируются более или менее аналогичные функциональные требования к системе, поэтому логично выделить их в отдельный класс пользователей, назвав его, скажем, *сотрудники, принимающие заявки.* Это может быть специалист по кредитованию, вице-президент, менеджер по работе с клиентами и даже банковский кассир —

при условии, что все они используют систему, чтобы помочь кому-либо подать заявку на получение займа.

Одни классы пользователей для вас важнее других. Когда вы принимаете решения о приоритетах или пытаетесь найти компромисс требований, выдвигаемых различными классами пользователей, мнение привилегированных классов имеет первостепенное значение. К последним относятся группы пользователей, работа которых с продуктом определяет, способствует ли он достижению заявленных бизнес-целей или нет. Это не означает, что заинтересованных лиц, оплачивающих разработку системы (они, вполне вероятно, вообще не являются ее пользователями), или тех, кто имеет большое политическое влияние, следует обязательно включать в привилегированные классы. Непривилегированные классы составляют те пользователи, которые по причинам безопасности, конфиденциальности или правовым причинам не работают с продуктом (Gause и Lawrence, 1999). Остальные классы пользователей можно проигнорировать. Они получат то, что получится, то есть при разработке системы вам не надо учитывать их интересы. Мнение прочих классов пользователей при определении требований к продукту имеют примерно одинаковое значение.

У каждого класса пользователей есть свой набор требований, сформировавшийся в ходе выполнения задач. Кроме того, появляются различные нефункциональные требования, например удобство применения, которые влияют на проектирование пользовательского интерфейса. Новичков волнует, насколько легко научиться работать (или вспомнить принципы работы) с продуктом. Такие клиенты предпочитают графические интерфейсы, упорядоченное представление информации на экране, подробные подсказки, мастеров и согласованность с другими приложениями, с которыми они уже знакомы. Тех, кто поопытнее, волнует легкость использования и эффективность работы. Они ценят клавиатурные сокращения, макросы, возможности настройки, панели инструментов, возможности создания сценариев и в некоторых случаях даже предпочитают интерфейс командной строки графическому интерфейсу пользователя.

Ловушка

Не упустите из виду классы косвенных или вторичных пользователей, Они могут обращаться к вашему приложению не напрямую, а работать с его данными и сервисами через другие приложения или отчеты. Однако даже опосредованный клиент все равно остается вашим клиентом.

Это может показаться странным, однако классы пользователей не обязательно состоят из людей. В качестве дополнительных классов пользователей можно рассматривать сторонние приложения и аппаратные компоненты, с которыми взаимодействует ваша система. Например, система впрыска топлива в автомобиле представляет собой пользовательский класс ПО, встроенного в систему управления двигателем. Система впрыска топлива не может говорить сама за себя, поэтому аналитику придется выяснить у инженера, разработавшего систему впрыска, требования к ПО, которое управляет процессом.

В самом начале работы над проектом необходимо определить и охарактеризовать различные классы пользователей, чтобы узнать у представителей всех важных классов их требования. Один из полезных способов определения классов называется ***«от расширения — к сжатию»*** («Expand Then Contract") (Gottesdiener, 2002). Для начала придумайте как можно больше классов пользователей: столько, сколько сможете. Не бойтесь, если их окажется несколько дюжин — позже вы объедините их и классифицируете. Важно не пропустить какой-либо класс, иначе это аукнется вам позже. Следующий этап — выявить группы с похожими потребностями: их можно объединить в один класс или рассматривать как несколько подклассов одного крупного класса пользователей. Постарайтесь, чтобы список отдельных классов не превышал пятнадцати.

Одна компания, разрабатывавшая ПО для примерно 65 корпоративных клиентов, рассматривала каждого из них как отдельного пользователя со своими потребностями. Классификация клиентов по шести классам значительно упростило работу с требованиями для

будущих версий продукта. Помните: не все заинтересованные в проекте лица на самом деле будут работать с продуктом, поэтому класс пользователей — это лишь группа людей, которым следует предоставить возможность выразить свое мнение о создаваемом продукте.

Таблица 6-1. Классы пользователей системы контроля химикатов

Химики (привилегированный класс)	Примерно 1000 химиков, работающие в шести зданиях, посредством системы запрашивают химикаты у поставщиков и со склада. Каждый химик использует систему несколько раз в день, преимущественно для запроса химикатов и контроля за контейнерами, поступающими в лабораторию и отправляемыми из нее. Химикам необходима возможность искать в каталогах поставщиков специальные химические структуры, импортированные из специальных утилит, для рисования таких структур
Отдел закупок	Около пяти сотрудников отдела закупок обрабатывают запросы на химические вещества, поступающие от других специалистов. Они размещают и отслеживают выполнение заказов поставщиками. Они не очень-то разбираются в химии, главное, что им нужно, чтобы поиск в каталогах был максимально простым. Специалистам отдела закупок не пригодятся возможности отслеживания контейнеров, предоставляемые системой. Каждый из них обращается к системе примерно 20 раз в день
Склад химикатов	Персонал склада химикатов - это шесть техников и один контролер, управляющий запасом из более чем 500 000 химических контейнеров. Они обрабатывают запросы от химиков на поставку контейнеров с трех складов, запросы поставщикам на новые химикаты и контролируют поступление контейнеров на склады и их отгрузку. Сотрудникам склада необходима только функция отчета о складских запасах. Из-за большого объема транзакций эта функция должны быть эффективной и автоматизированной
Отдел охраны труда и техники безопасности (привилегированный класс)	Специалисты отдела охраны труда и техники безопасности с помощью системы генерируют ежеквартальные отчеты в соответствии с местным и федеральным постановлениям об использовании и утилизации химических веществ. Скорее всего менеджеру этого отдела придется несколько раз в год запрашивать новую форму отчетов, которая зависит от последних правительственных постановлений. Эти запросы об изменениях имеют высочайший приоритет и должны быть реализованы в самые короткие сроки

Задокументируйте классы пользователей и их отличительные черты, меру ответственности и физическое расположение в спецификации требований к ПО. Менеджер проекта по разработке системы контроля химикатов, о которой шла речь в предыдущих главах, выявил следующие классы пользователей и их характеристики (табл. 6-1), Включите в спецификацию требований к ПО всю существенную информацию о каждом классе пользователей, например относительный или абсолютный размер и привилегированность класса. Это поможет команде разработчиков определить, какие запросы об изменениях наиболее важны, и в дальнейшем оценить ценность изменений. Примерная оценка объема и типа системных транзакций позволит специалисту по тестированию разработать алгоритм использования системы и в последующем планировать действия по проверке системы.

Чтобы было легче внедрить идею с классами пользователей в жизнь, стоит описать

типичного представителя каждого класса (Cooper, 1999), например, вот так:

Фред, 41 год, работает химиком в Contoso Pharmaceuticals уже 14 лет, с тех пор как получил степень кандидата наук. Нетерпелив с компьютерами. Обычно Фред одновременно работает над двумя проектами в смежных областях химии. В его лаборатории хранится около 400 контейнеров и баллонов с химикатами. Ежедневно ему требуется в среднем 4 новых химиката со склада. Два из них — промышленные химикаты из имеющихся на складе, один необходимо заказать и еще один придется взять из химических образцов компании. Иногда Фреду требуются опасные химикаты, для работы с которыми необходима специальная подготовки. Приобретая какой-либо химикат впервые, Фред хочет, чтобы ему по электронной почте автоматически поступали соответствующие материалы по технике безопасности. Ежегодно Фред создает около 10 новых химикатов, которые отправляет на склад. Он хочет получать по электронной почте автоматически сгенерированный отчет о заказанных им в прошлом месяце химикатах; это позволит ему контролировать расход химических веществ.

Изучая требования химиков, рассматривайте Фреда как архетип данного класса пользователей и задайте себе вопрос: «Что Фреду нужнее всего?»

Представители пользователей

При разработке каждого типа проекта, включая корпоративные информационные системы, коммерческие приложения, комплексные решения, интегрированные системы, встроенные системы, интернет приложения и ПО, конструируемые на заказ, необходимо привлечь с работе представителей пользователей, выражающих мнение большей части клиентов. Позаботьтесь, чтобы они участвовали во всех этапах, а не только в одном из начальных — формулировании требований. Ну и, конечно, должны быть представлены все классы пользователей.

Проще всего наладить контакт с пользователями, когда вы создаете внутрикорпоративные приложения. Если вы разрабатываете коммерческое ПО, можно расспросить тех, кто занимается бета-тестированием продукта или «обкаткой» сайта, чтобы выяснить у них требования на начальных этапах работы (подробнее — в разделе «Сторонники продукта, приглашенные со стороны» далее в этой главе). Иногда полезно создать фокус-группы из пользователей для анализа ваших продуктов или ПО конкурирующих компаний.

Некая компания предложила участникам фокус-группы протестировать различные цифровые фотоаппараты и компьютеры. Как показали результаты, ПО фотоаппаратов этой компании слишком долго выполняло самую обычную операцию из-за того, что в архитектуру были заложены наименее вероятные варианты использования. После внесения изменений в следующую версию фотоаппарата количество жалоб клиентов на малую скорость работы уменьшилось.

Убедитесь, что в фокус-группе представлены все классы пользователей, потребности которых должны определять возможности продукта, а также люди с различным опытом: и эксперты, и новички. Если в фокус-группу входят только приверженцы последних достижений и любители помечтать, вас захлестнет вал сложных и технически трудных требований, которые могут оказаться неинтересными большей части вашей целевой аудитории.

На рис. 6-2 показаны некоторые типичные пути распространения информации то есть, как голос пользователя достигает ушей разработчика. Как отмечалось в одном исследовании, в наиболее успешных проектах всегда больше каналов общения вообще и тех, что напрямую связывают разработчика и клиента, чем в менее успешных проектах (Keil и Carmel, 1995). Лучше всего, когда разработчики имеют возможность сами пообщаться с нужными пользователями; то есть выполняют роль аналитиков. Как и в детской игре «Испорченный телефон», дополнительные промежуточные звенья увеличивают вероятность непонимания. Например, требования, озвученные менеджером конечных пользователей, скорее всего, не совсем точно передают реальные потребности этих пользователей.

Тем не менее некоторые из этих промежуточных звеньев полезны, например, когда опытный аналитик требований работает с пользователями или другими участниками, собирая, оценивая, уточняя и организуя их информацию. Оцените возможные риски, используя в качестве

представителей реального мнения клиента персонал отдела маркетинга, менеджеров продукта, профильных специалистов или каких-либо других лица. Несмотря на препятствия и затраты, связанные с поиском и выбором представителей пользователей, ваш продукт и клиенты пострадают, если вы пренебрежете общением с людьми, способными предоставить наиболее точную информацию.

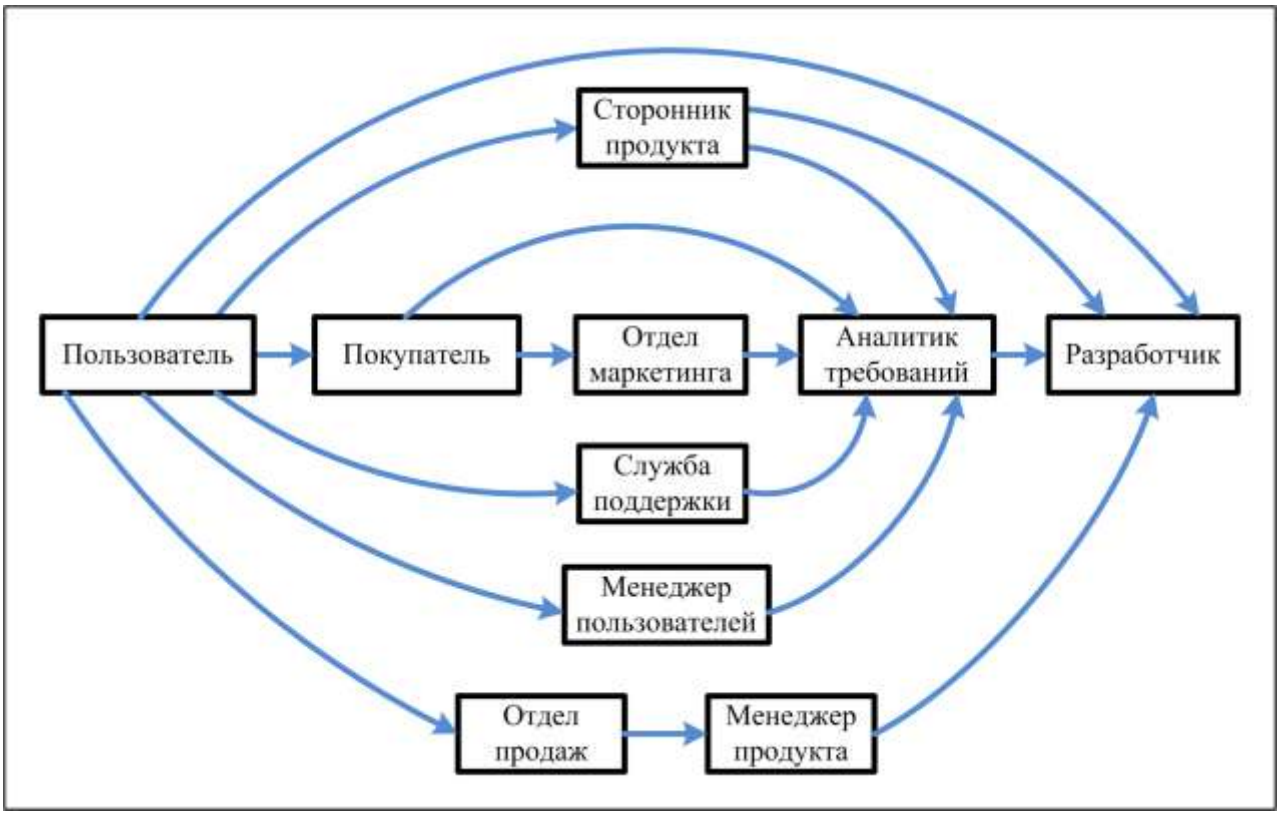


Рис. 6-2 Некоторые возможные пути распространения информации от пользователей к разработчикам

Сторонники продукта

Много лет назад я работал в небольшой группе разработки программных продуктов, которая обеспечивала поддержку научных исследований в крупной корпорации. Во всех наших проектах принимали участие несколько наиболее активных представителей пользователей, которые помогали формулировать требования. Мы называли их ***сторонниками продукта*** (product champion) (или *приверженцами проекта,* хотя этот термин относится скорее к тем, кто управляет финансовой стороной проекта) (Wiegers, 1996a). Привлечение искренне заинтересованных в продукте пользователей — это эффективный способ структурировать партнерство клиентов и разработчиков.

> **Ловушка**
> Опасайтесь менеджеров и разработчиков, считающих, что они и без общения с пользователями уже все знают о потребностях последних.

Каждый сторонник продукта — это основной посредник между членами отдельного класса пользователей и аналитиком требований. В идеале сторонники должны быть реальными пользователями, а не псевдопользователями, как, например спонсоры, покупатели, специалисты по маркетингу, менеджеры или разработчики ПО, которые представляют себя на месте пользователя. Их задача — выяснить потребности остальных членов класса пользователей, который они представляют, и согласовать их. Таким образом, разработка требований — это общее дело

аналитика и выбранных клиентов; хотя документацию составляет все-таки аналитик.

Лучшие из сторонников продукта имеют четкое представление о новой системе и в полной мере увлечены ее, так как понимают, какие преимущества она даст им и их коллегам. Сторонник продукта должен уметь отлично общаться и пользоваться авторитетом в компании. Кроме того, необходимо, чтобы он разбирался в предметной области и рабочей среде приложения. Такие сотрудники весьма заняты своей основной работой, поэтому вам необходимо продумать очень убедительные доводы, дабы обосновать крайнюю важность именно этих людей для успеха проекта. Моя команда и я убедились, что хорошие сторонники сделали очень много для успеха наших проектов, и мы с удовольствием выражаем им нашу признательность за их вклад.

Мы рады отметить и еще одно преимущество от сотрудничества со сторонниками продукта. В нескольких наших проектах участвовали отличные ребята, которые от нашего лица общались с коллегами и клиентами, когда те возмущались тем, что проект до сих пор не завершен. «Не волнуйтесь, — говорили они коллегам и начальству. — Мы понимаем и принимаем то, как команда разработчиков подходит к делу. Время, которое мы потратим сейчас на работу над требованиями, поможет нам получить именно ту систему, которую мы хотим, и сэкономит время в дальнейшем. Не беспокойтесь». Такое сотрудничество помогает устранить напряжение, зачастую возникающие между клиентами и командой разработчиков.

Наилучшие результаты отмечены, если каждый сторонник продукта обладает всеми полномочиями для принятия необходимых решений от имени класса пользователей, который он представляет. Если решения сторонника продукта регулярно отвергаются менеджерами или группой разработчиков, он попусту тратит свое время и добрую волю. Тем не менее горячим сторонникам продукта следует помнить, что они — не единственные клиенты. Проблемы возникают, если тот, кто играет такую важную роль посредника, общается с коллегами не должным образом и выражает только собственные пожелания и идеи.

Сторонники продукта, приглашенные со стороны

При разработке коммерческого ПО иногда трудно найти сторонников продукта вне компании. Если у вас налажены тесные деловые отношения с какими-либо крупными корпоративными клиентами, они могут благосклонно отнестись к возможности поучаствовать в формировании требований. Сторонников продукта, приглашенных со стороны, можно заинтересовать экономически, например предложить им скидку на продукт или оплатить их работу над требованиями. Тем не менее, здесь существует некоторая опасность: вам придется научиться слушать не только ярых приверженцев, чтобы не упустить мнения других клиентов. При наличии разнородной клиентской базы сначала нужно определить основные требования, общие для всех клиентов, затем дополнительные требования, важные для отдельных клиентов, сегментов рынка или классов пользователей.

В некоторых случаях компании-разработчики коммерческого ПО полагаются на собственных профильных специалистов или внешних консультантов, подменяющих реальных пользователей, о которых может быть ничего не известно или которых трудно привлечь к работе над, проектом. Другой вариант — нанять подходящего сторонника продукта с соответствующим опытом. Одна компания, создававшая систему с центральным сервером и клиентскими приложениями для точек розничной торговли, пригласила в качестве сторонников продукта на полный рабочий день трех менеджеров магазина. Вот еще один пример: Арт, мой давний семейный доктор, оставил врачебную практику и устроился в компанию, разрабатывающую медицинское ПО, где представлял мнение и интересы врачей. Работодатель Арта был уверен: штатный сотрудник, имеющий врачебный опыт, поможет компании разрабатывать ПО, которое придется по душе другим докторам. Другая компания пригласила на работу нескольких бывших сотрудников одного из своих основных клиентов. Эти люди поделились ценным опытом в предметной области и описали политику организации-клиента.

Если сторонник продукта — бывший пользователь или увлеченно играет роль пользователя, таковым не являясь, опасайтесь, что его восприятие проблем и потребности реальных пользователей могут различаться. Некоторые предметные области меняются очень быстро, другие относительно стабильны. Медицина развивается очень быстро, тогда как многие корпоративные

бизнес-процессы остаются неизменными в течение лет. Основной вопрос в том, сможет ли сторонник продукта, независимо от его опыта и настоящей должности, точно отражать потребности реальных пользователей,

Чего следует ожидать от сторонника продукта

Чтобы сотрудничество со сторонниками продукта оказалось успешным, задокументируйте, что именно, по-вашему, они должны делать. Это поможет вам полнее выразить ваши ожидания от участия в проекте конкретного человека. В табл. 6-2 перечислены некоторые обязанности сторонника продукта. Не каждый станет выполнять их все; используйте эту таблицу как отправную точку для обсуждения с каждым сторонником продукта его обязанностей.

Таблица 6-2. Возможные обязанности сторонника продукта

Категория	• Действия
Планирование	• Уточнение рамок и ограничение возможностей продукта • Определение интерфейсов для работы с другими системами • Оценка влияния новой системы на бизнес-операции • Разработка путей перехода со старых приложений на новые
Требования	• Сбор требований от других пользователей • Разработка сценариев и вариантов использования продукта • Разрешение конфликтов между высказанными требованиями • Определение приоритетов выполнения • Определение атрибутов качества и производительности • Оценка прототипов пользовательского интерфейса
Проверка и подтверждение	• Изучение документов с требованиями • Определение критериев приемлемости продукта для пользователей • Разработка вариантов тестирования на основе сценариев использования • Создание наборов тестовых данных • Бета-тестирование
Помощь пользователям	• Написание фрагментов руководств пользователя и справочных систем • Подготовка материалов для учебных пособий • Демонстрация продукта коллегам
Управление изменениями	• Оценка недостатков и определение порядка их устранения • Оценка предложений об усовершенствовании системы и определение порядка их реализации • Оценка того, как предполагаемые изменения требований повлияют на пользователей и на бизнес-процессы • Участие в принятии решений о внесении изменений

На что способны несколько сторонников продукта

Один человек вряд ли сможет описать потребности всех пользователей приложения. Для системы контроля химикатов определено четыре основных классов пользователей, и поэтому для нее следует выбрать четырех сторонников продукта из числа сотрудников Contoso Pharmaceuticals. На рис. 6-3 показана структура команды, которую менеджер проекта организовал из аналитиков и

сторонников продукта для сбора корректных требований из корректных источников. Сторонники не привлекались на полный рабочий день, каждый из них уделял проекту несколько часов в неделю. Три аналитика совместно с четырьмя сторонниками выявляли, анализировали и документировали их требования (один из аналитиков работал с двумя сторонниками, поскольку классы пользователей «Отдел закупок» и «Отдел охраны труда и техники безопасности» невелики и выдвинули всего несколько требований. Один из аналитиков сводил все требования в единую спецификацию требований к ПО.

Один человек не может в полной мере представить все разнообразные требования крупного класса пользователей, например нескольких сотен химиков Contoso. Чтобы помочь ему, сторонник продукта от класса «Химики» собрал резервную команду из пяти химиков, работающих в других подразделениях компании. Эти сотрудники представляют подклассы обширного класса пользователей «Химики». Такой иерархический подход позволил вовлечь в разработку требований дополнительных пользователей и в то же время сэкономить на проведении множества круглых столов и дюжин интервью. Сторонник продукта от класса «Химики», Дон, всегда стремился к консенсусу, но в тех случаях, когда соглашение достичь не удавалось, он без колебаний принимал необходимые решения, чтобы работа не буксовала. Необходимость в резервной команде отпадает, если класс пользователей невелик или сплочен настолько, что интересы группы может в полной мере представлять один человек.

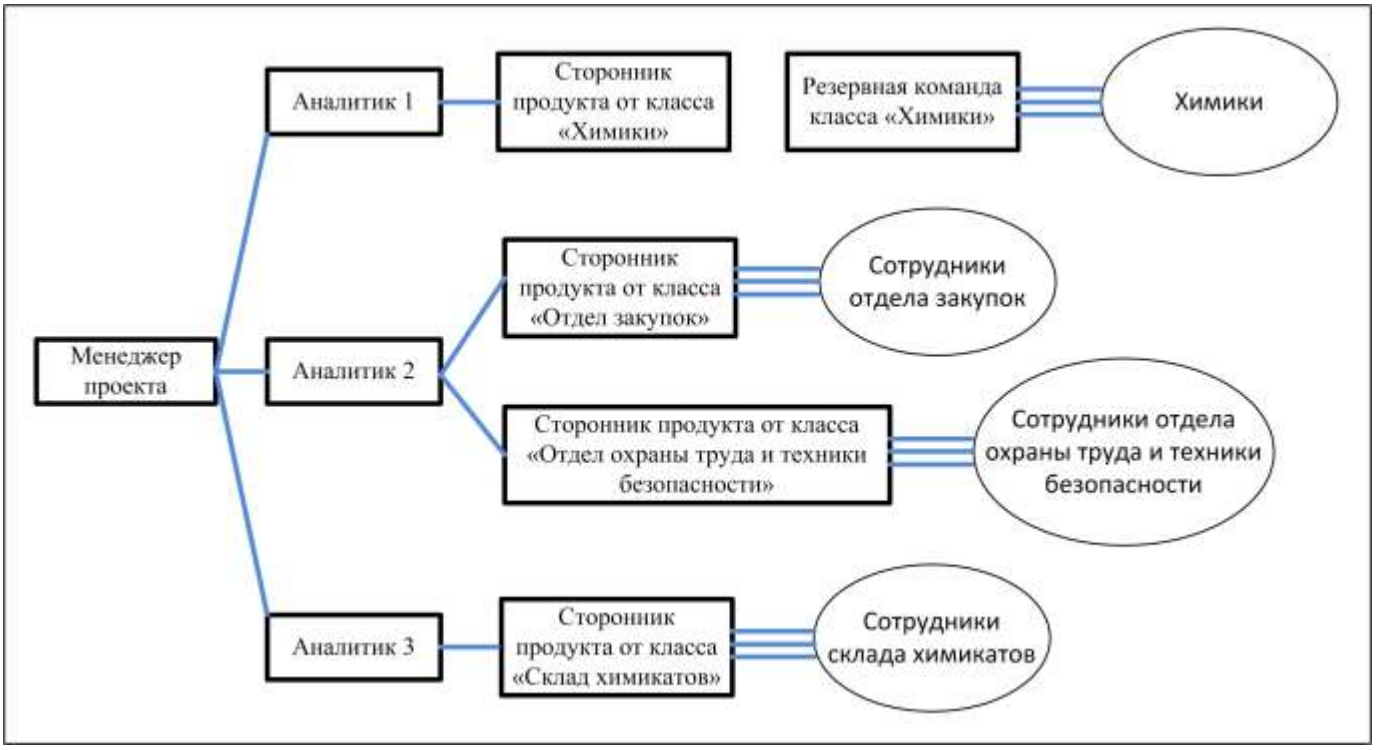


Рис. 6-3. Структура команды из аналитиков и сторонников продукта для системы контроля химикатов

Безголосый класс пользователей

Аналитик требований в компании Humongous Insurance был очень доволен, узнав, что пользующийся авторитетом сотрудник, Ребекка, согласилась стать сторонником продукта для новой системы обработки исков. У Ребекки было много идей насчет возможностей и пользовательского интерфейса системы. Вдохновленная возможностью работать под руководством эксперта, команда разработчиков с удовольствием реализовала ее предложения. Однако, выпустив продукт, они были шокированы количеством жалоб на трудности работы с системой.

Ребекка — опытный пользователь. Предложенные ею функции, обеспечивающие удобство работы с системой, хороши для таких же опытных пользователей, как и она сама, однако 90% клиентов, гораздо менее опытных, нашли, что система интуитивна непонятна и трудна для освоения. Аналитик требований вовремя не понял, что у системы обработки исков по крайней мере два класса пользователей. Большая группа начинающих пользователей оказалась лишенной права на участие в разработке требований и пользовательского интерфейса, поэтому компании Humongous пришлось потратиться на переделку проекта, а она обошлась дорого. Аналитику следовало привлечь второго сторонника продукта — выразителя интересов большого класса неопытных пользователей.

Как «продать» идею о необходимости привлечения сторонника продукта

Будьте готовы встретить сопротивление, когда вы предложите привлечь к работе над проектом сторонников продукта. «Пользователи слишком заняты». «Менеджеры хотят сами принимать решения». «Они затормозят нашу работу». «Мы не можем себе этого позволить». «Я не знаю, что могу предложить в качестве сторонника продукта». Некоторые пользователи не хотят высказывать свои требования к системе, которая, как они думают, заставит их изменить методы работы или создаст угрозу увольнения. Иногда менеджеры неохотно делегируют обычным пользователям полномочия, касающиеся требований.

Отделив бизнес-требования от требований пользователей, вы смягчите эти препятствия. Будучи реальным пользователем, сторонник продукта принимает решения на уровне интересов пользователей в рамках, налагаемых бизнес-требованиями проекта. Финансовый менеджер сохраняет всю полноту власти, чтобы принимать решения, касающиеся вида продукта, его границ, графика работы и затрат. Документируя и обсуждая роль и обязанности апологета продукта, вы даете предполагаемым кандидатам на эту роль возможность уверенно отвечать на вопрос, чем же они занимаются.

Если же вы столкнетесь с сопротивлением, укажите, что недостаточное участие пользователей — общепризнанная основная причина провала проектов по разработке ПО (The Standish Group, 1995). Напомните оппонентам о проблемах предыдущих проектов, вызванных недостаточным вовлечением пользователей в работу. Каждая компания хранит свои страшилки о новых системах, которые не удовлетворили потребности пользователей, или оказались не столь полезны, или не оправдали ожидания. Вы не можете позволить себе переделывать системы или отказываться от них, если они не соответствуют необходимым требованиям, только потому, что никто не разобрался в требованиях. Привлечение сторонников продукта уменьшает такой риск.

В какие ловушки можно угодить, привлекая сторонников продукта

Модель привлечения сторонника продукта хорошо зарекомендовала себя во многих рабочих средах. Она имеет смысл, только если сторонник понимает и принимает свои обязанности,

обладает полномочиями для принятия решений на уровне требований пользователей и имеет время для выполнения данной работы. Остерегайтесь следующих потенциальных проблем.

- Некоторые менеджеры изменяют решения, принятые квалифицированным и официально назначенным сторонником продукта. Возможно, у менеджера в последнюю минуту появилась новая безумная идея, он из прихоти захотел изменить направление работы или думает, что знает все потребности пользователей. Зачастую результат такого поведения — недовольные пользователи, срыв сроков сдачи проекта из-за того, что разработчики пытаются реализовать последние замечания менеджеров, а также расстроенные сторонники продукта, чувствующие, что руководство им не доверяет.
- Сторонник продукта, забыв, что он представляет других клиентов, озвучивает только собственные требования: конечно же, ему не удастся хорошо выполнять свои обязанности. Возможно, лично ему результат понравится, а вот остальным — вряд ли.
- Сторонник продукта, не имеющий четкого представления о новой системе, может уступить решение важных вопросов аналитику. Если все идеи аналитика вызывают одобрение сторонника, его трудно назвать полноценным помощником.
- Из-за нехватки времени опытный пользователь назначает сторонником продукта менее квалифицированного коллегу. Это может привести к давлению со стороны опытного пользователя, который по-прежнему желает направлять ход работы над проектом.
- Остерегайтесь пользователей, пытающихся выступить от имени класса, к которому они не относятся. В случае с системой контроля химикатов компании Contoso сторонник продукта от класса «Сотрудники склада химикатов настаивала, что может выразить требования класса пользователей «Химики». Ее было весьма трудно убедить, что это не его работа, однако аналитик не позволил ей оказывать давление. Менеджер проекта пригласил сторонника от класса «Химики», и тот проделал блестящую работу по сбору, оценке и передаче требований этих специалистов.

Кто принимает решения

Кто-то должен устранять конфликты между требованиями разных классов пользователей, сглаживать противоречия и решать возникающие вопросы относительно границ проекта. На ранних стадиях проекта необходимо определить тех, кто будет принимать решения, касающиеся требований. Если не ясно, кто за это отвечает, или уполномоченные лица отказываются брать на себя такую ответственность, обязанность принимать решения по умолчанию возлагается на разработчиков. Это — неудачный выход, поскольку у разработчиков обычно нет необходимых знаний, опыта и стратегического видения для принятия оптимальных бизнес-решений.

Не существует единственно верного ответа на вопрос о том, кто должен принимать решения по требованиям к программному проекту. Аналитики иногда прислушиваются к наиболее громогласному сотруднику или мнению начальства. Это понятный, но не лучший выбор. По возможности решения должны принимать сотрудники на низшем уровне организационной иерархии, которые непосредственно занимаются выпуском продукта и знают все о нем. Прежде чем приниматься за выработку первого решения, каждая группа должна определить соответствующее ***правило решения*** (decision rule) — согласованный метод принятия решения (Gottesdiener, 2001). Лично мне нравится совместное принятие решений или принятие решений в ходе совещания (когда собираются идеи и мнения множества заинтересованных лиц), а не принятие решений через достижение консенсуса всех заинтересованных лиц. Последний вариант - идеальный, однако вы не можете постоянно тормозить работу, ожидая, пока все заинтересованные лица согласуют каждый вопрос.

Ниже описаны некоторые конфликты, которые могут возникнуть в ходе работы над проектами, а также способы их устранения. Ведущим проект сотрудникам необходимо определить, кто будет принимать решения при возникновении подобных ситуаций, кто обладает правом голоса, если согласие не достигнуто, и кто будет в дальнейшем санкционировать решения по важнейшим вопросам.

- Если отдельные пользователи не могут прийти к согласию при определении требований, решение принимают сторонники продукта. Суть привлечения апологетов такова: они уполномочены и обязаны устранять конфликты требований, возникающие у тех, кого они представляют.
- Если требования различных классов пользователей или сегментов рынка несовместимы, сделайте выбор в пользу наиболее важного класса пользователей или сегмента, который окажет наибольшее влияние на успех продукта на рынке.
- Иногда все корпоративные клиенты одновременно требуют, чтобы архитектура проекта удовлетворяла именно их требования. В таком случае опять-таки на основе бизнес-целей проекта определите, кто из клиентов оказывает наиболее сильное влияние на успех или провал проекта,
- Иногда требования, высказываемые менеджером пользователей, не совпадают с интересами реальных пользователей. И хотя требования последних должны соответствовать бизнес-требованиям, менеджеры, которые не входят в конкретный класс пользователей, обязаны прислушаться к стороннику продукта, выражающему мнение соответствующего класса.
- Если представление разработчиков о создаваемом продукте не совпадает с пожеланиями клиентов, последнее слово за клиентами.

Ловушка

Не оправдывайте любые запросы клиента только потому, что «клиент всегда прав». Клиент прав не всегда. Однако у него есть своя точка зрения, и команда разработчиков должна понимать и уважать ее.

- Аналогичная ситуация возникает, если требования маркетологов или менеджеров по продукту конфликтуют с представлением разработчиков о продукте. Мнение маркетологов, как представителей клиента, более весомо. Однако я знаю случаи, когда они потакали клиенту в его любых желаниях, как бы невыполнимы и дороги те ни были. Я также видел и обратные примеры, когда маркетологи предоставляли так мало информации, что разработчикам приходилось самим определять архитектуру продукта и формулировать требования.

Переговоры о требованиях не всегда проходят так, как они должны проходить по мнению аналитиков. Клиент может отвергать все предложения разумных альтернатив и другие точки зрения. Команде следует определить, кто будет принимать решения по требованиям к проекту, еще до начала подобных столкновений. В противном случае нерешительность и пересмотр уже принятых решений приведут к тому, что работа над проектом заглохнет в бесконечных спорах.

Что теперь?

Сравните схему на рис. 6-1 с тем способом, посредством которого вы узнаете мнение клиентов. Возникают ли у вас проблемы с распространением информации? Определите кратчайшие и наиболее подходящие пути распространения информации, которые пригодятся для сбора требований в будущем.

Определите для своего проекта различные классы пользователей. Какие них привилегированные, а какие (если таковые имеются) — нет? Кто бы мог стать хорошим сторонником продукта для каждого из важных классов пользователей?

Используя табл. 6-2 в качестве отправной точки, определите обязанности сторонника продукта. Обсудите конкретные пункты с каждым предполагаемым сторонником продукта и его менеджером.

Определите, кто должен принимать решения по вопросам, связанным с требованиями к проекту. Насколько эффективен процесс принятия решений, используемый в настоящее время? Где он дает сбой? Правильно ли выбраны лица, принимающие решения? Если нет, кто должен принимать решения? Предложите лицам, принимающим решения, последовательность действий для достижения согласия по вопросам требований

Глава 7 Как услышать голос клиента

«Доброе утро, Мария. Я — Фил, аналитик требований к новой информационной системе для сотрудников, которую мы собираемся разрабатывать для вас. Спасибо, что согласились стать сторонником продукта этого проекта. Ваша информация нам очень пригодится. А теперь расскажите мне, что же вам все-таки нужно?»
«Гм, что же мне нужно, — размышляет Мария. — Не представляю, с чего и начать. Ну, новая система должна работать намного быстрее предыдущей. Потом, вы же знаете, как старая система зависает, если имя какого-либо сотрудника оказывается слишком длинным — нам приходится обращаться в службу поддержки, чтобы они ввели имя. В новой системе длинные имена должны обрабатываться без сбоев. Да, согласно новому закону, нам нельзя использовать номера социального страхования в качестве идентификаторов сотрудников, и с вводом новой системы в эксплуатацию нам придется заменить все эти данные. Новые идентификаторы предполагается сделать шестизначными числами. Также пригодилась бы функция отчета о времени, затраченном каждым сотрудников в текущем году на обучение. А еще мне хотелось бы получить возможность менять фамилию сотрудников, даже если их семейное положение осталось прежним».
Фил добросовестно записал все пожелания Марии, но его голова пошла кругом. Он не знал, что делать с этой информацией, и не имел ни малейшего представления, что же передать разработчикам. «Ну что ж, — подумал он, — если Мария хочет, чтобы мы это сделали, полагаю, нам придется это сделать».

Сердце процесса формулирования требований — их ***выявление*** (elicitation), когда потребности и ограничения высказывают лица, заинтересованные в проекте. Цель — обозначить требования пользователей, которые занимают средний уровень в трехступенчатой модели требований к ПО (подробнее об этом в главе 1, два других уровня — бизнес-требования и функциональные требования). Это задачи, которые пользователям необходимо выполнять средствами системы, а также представления пользователей об эффективности, удобстве и простоте работы с системой и других атрибутах качества системы. В этой главе обсуждаются общие принципы эффективного выявления требований.

Аналитику необходима структура для систематизации массива информации, полученной в процессе выявления требований. Просто задавая пользователям вопрос: «Чего вы хотите?», вы получите массу неупорядоченной информации, в которой легко утонуть. Вопрос: «Что вам требуется делать?» — гораздо лучше. В главе 8 рассказывается как с помощью таких приемов, как варианты использования продукте или построение таблицы откликов на события, создать структуру для упорядочения пользовательских требований.

Результат этапа формулирования требований — согласованное представление о потребностях всех заинтересованных в проекте лиц. Теперь разработчикам гораздо легче предлагать альтернативные способы удовлетворения этих потребностей. Те, кто занимается выявлением требований, не должны поддаваться искушению немедленно приступить к проектированию системы — до того момента, как проблема станет абсолютно ясной. В противном случае готовьтесь к значительной переработке проекта по мере детализации требований, Концентрация на задачах пользователей, а не на интерфейсе, внимание к ключевым потребностям, а не к пожеланиям позволяет команде не отвлекаться на детали архитектуры — сейчас это несколько преждевременно.

Для начала подумайте, как вы собираетесь выявлять требования к проекту. Даже простой план повышает шансы на успех и делает более реалистичными ожидания всех заинтересованных лиц. Только четко сформулировав потребности, касающиеся ресурсов, графика и выпуска, можно избежать того, что лиц, участвующих в выявлении требований, отзовут для исправления ошибок или выполнения другой работы. План должен отражать:

- цели выявления требований (например, проверка рыночных данных, исследование вариантов использования или разработка подробного набора функциональных

требований к системе);

- стратегии и способы выявления требований (например, сочетание опросов, семинаров, встреч с клиентами, интервью и других приемов с возможным использованием различных подходов для разных групп заинтересованных лиц);
- результаты выявления требований (например, перечень вариантов использования продукта, подробная спецификация требований к программному обеспечению, анализ результатов опроса или спецификация атрибутов качества и производительности);
- график и смету ресурсов (определите, кто из разработчиков и клиентов будет участвовать в различных операциях по выявлению требований; примерно оцените усилия и время на выявление требований);
- риски, связанные с выявлением требований (укажите факторы, которые могут нарушить график работ по выявлению требований, оцените опасность каждого фактора и решите, как смягчить его или управлять им).

Выявление требований

Возможно, выявление требований — самый трудный, важный, подверженный ошибкам и требующий интенсивного общения этап разработки ПО. Как вы уже знаете из главы 2, эта процедура может оказаться успешной только при условии сотрудничества клиентов и разработчиков. Аналитик должен создать атмосферу, благоприятную для тщательного исследования обусловленного продукта. Чтобы облегчить общение, используйте лексику предметной области, а не прессингуйте клиентов компьютерным жаргоном. Не полагайтесь на то, что все участники одинаково понимают важные термины предметной области, создайте словарь. Однако объясните клиентам, что обсуждение возможной функциональности — это еще не обязательство включить ее в продукт. Этот этап обособлен от анализа приоритетов, осуществимости и ограничений, налагаемых реальностью. Заинтересованным в проекте лицам необходимо расставить приоритеты в списке «голубых грез», чтобы проект не стал рыхлым и бесполезным.

Мастерство управления дискуссией по выявлению требований приходит с опытом, очень помогают в таком деле тренинги, где слушатели учатся брать интервью, общаться в группах, разрешать конфликтные ситуации и т.д. Как аналитику, вам придется «нырнуть» в поток требований клиента и, «опустившись поглубже», разобраться в его истинных потребностях. Просто задав несколько раз вопрос типа «Почему?», вы сможете перейти от обсуждения уже существующего решения к выявлению проблемы. Вопросы, допускающие несколько вариантов ответа, помогут вам разобраться в бизнес-процессах и понять, как новая система повысит их эффективность. Поинтересуйтесь, какие задачи собираются выполнять пользователи, и как они будут работать с системой. Вообразите, что вы обучаетесь служебным обязанностям пользователя или непосредственно выполняете их под его руководством. Какие задачи вам придется решить? Какие вопросы у вас возникают. Другой подход — влезть в «шкуру» новичка, которого обучает опытный пользователь. Вы задаете массу вопросов, а он управляет беседой, выбирая важную, с его/ее точки зрения, тему для обсуждения.

Попробуйте метод исключений. Что мешает пользователю успешно выполнить задачу? Как система должна реагировать на различные ошибочные условия? Задавайте вопросы, начинающиеся со слов: «А что еще могло бы...», «Что произойдет, когда...», «Вам когда-нибудь требовались...», «Где вы получаете...», «Почему вы делаете (или не делаете)...» и «А кто-нибудь когда-либо...». Задокументируйте источник каждого требования, чтобы, если потребуется, понять их причину и проследить, на каких же потребностях клиентов основываются те или иные направления разработки.

Когда вы разрабатываете следующую версию системы, попросите пользователей припомнить три вещи, которые раздражают их сейчас больше всего. Этот вопрос поможет вам понять, почему же систему следует менять. При этом также становится ясно, чего же ждут пользователи от новой системы. Как и при любой модификации, неудовлетворение настоящим дает отличную пищу для принятия нового.

Попытайтесь вытащить на «свет Божий» все предположения клиентов и разрешить конфликты. Читайте между строк, чтобы определить те возможности или характеристики, которые клиенты полагают само собой разумеющимися и даже не считают нужным обрисовать. Gause и Weinberg

(1989) предлагают использовать ***бесконтекстные вопросы*** *(context-freequestions),* узкоспециализированные вопросы и вопросы, допускающие разные толкования, чтобы выявить информацию о бизнес-проблемах и их возможных решениях. Реакция клиентов на такие вопросы, как «Какой уровень точности необходим в продукте?» или «Почему вы не согласны с ответом Мигеля?", иногда более действенны, чем вопросы с ответами типа «да/нет» или «A/B/C».

В процессе сбора информации работающий творчески аналитик не только фиксирует слова клиента, но и подкидывает пользователям новые идеи и предлагает альтернативы. Иногда пользователи не представляют, какие возможности готовы предоставить разработчики; они страшно обрадуются, если вы предложите функциональность, которая сделает систему особенно удобной. В тех случаях, когда пользователи не способны ясно выразить свои потребности, аналитику стоит понаблюдать за их работой, чтобы самому разобраться, какие операции следует автоматизировать. Отлично, когда аналитикам удается выйти за рамки, ограничивающие мышление людей слишком тесно связанных с конкретной предметной областью. Ищите удобные случаи повторно использовать функциональность, которая уже реализована в другой системе.

Интервью с отдельными потенциальными пользователями или с группами — традиционный источник информации при сборе требований, касающихся как коммерческих продуктов, так и информационных систем. [О проведении интервью с пользователями см. Beyer и Holtzblatt (1998), Wood и Silver (1995), а также McGraw и Harbison 97).] Вовлечение пользователей в процесс сбора информации — это способ получить поддержку, в том числе и финансовую, проекта. Попытайтесь понять, из чего следуют конкретные требования пользователей. Поэтапно рассмотрите работу пользователей и постарайтесь понять ее логику. Блок-схемы и деревья решений позволяют отобразить логические пути принятия решений. Убедитесь, что все понимают, почему система ***должна*** выполнять конкретные функции. Иногда предложенные требования отражают устаревшие или неэффективные бизнес-процессы, которые не следует встраивать в новую систему.

После каждого интервью документируйте элементы, обсуждавшиеся группой, и попросите интервьюируемых просмотреть список и внести исправления. Просмотр на ранних стадиях необходим для успешного сбора требований, поскольку только те люди, которые эти требования предоставили, могут судить, правильно ли они зафиксированы. В дальнейшем также не отказывайтесь от обсуждений — это позволит разрешить все противоречия и заполнить пробелы.

Польза семинаров

Аналитики требований часто проводят совместные семинары, упрощающие сбор информации о требованиях. Это позволяет весьма эффективно наладить связи между пользователями и разработчиками (Keil и Carmel 1995). Основная задача ответственного за мероприятие сотрудника — создать условия для успешного выполнения задачи; именно он планирует семинар, выбирает участников и следит, чтобы обсуждение проводилось продуктивно. Если вы собираетесь применять новые технологии сбора требований, ответственным за первый семинар следует назначить стороннего сотрудника — не из вашей команды. В этом случае аналитик сможет полностью сосредоточиться на обсуждении. Пригласите также секретаря, чтобы он фиксировал все идеи, возникающие в ходе обсуждения.

Согласно одному из источников, «Организация какого-либо мероприятия — искусство управления людьми в ходе этого мероприятия, которое позволяет достичь согласованных решений в атмосфере сотрудничества, высокопроизводительного труда и заинтересованности в результатах своей работы» (Sibbet, 1994). О семинарах, упрощающих сбор требований, подробно рассказано в труде Ellen Gottesdiener (2002) «Requirements by Collaboration» (Выявление требований с помощью сотрудничества). Gottesdiener описывает спектр приемов и средств для облегчения семинаров. Вот некоторые из них.

Определите основные правила. Участники должны договориться об основных правилах проведения семинаров (Gottesdiener, 2002). Например, таких:

- своевременно начинать и заканчивать семинар;
- не опаздывать после перерывов;
- не проводить несколько обсуждений одновременно;
- следить, чтобы каждый принимал участие в работе;

- комментировать и критиковать решение, а не личность.

Придерживайтесь границ проекта. Чтобы удостовериться, что предлагаемые пользовательские требования не выходят за текущие границы проекта, используйте документ об образе и границах проекта.

Следите, чтоб на каждом семинаре уровень обобщения соответствовал выбранным целям. Периодически участники могут углубляться в обсуждения несущественных деталей. На это уходит масса времени, которое на начальном этапе работы следует потратить на прояснение пользовательских требований — время деталей наступит позже. Задача организатора — по мере необходимости возвращать участников к теме обсуждения.

Ловушка

Избегайте преждевременного углубления в детали. Фиксируя мельчайшие подробности того, что люди уже понимают, вы не снизите риски, остающиеся из-за размытости требований.

Пользователи легко переходят к обсуждению того, где в отчете или диалоговом окне следует разместить элементы, даже еще до того, как разработчики согласятся с поставленными задачами. Если вы включите данные детали в требования, то в некоторой степени ограничите процесс разработки. К разработке пользовательского интерфейса следует приступать позднее, хотя предварительные наброски интерфейса полезны на любом этапе при иллюстрации возможных способов реализации требований. Проверка осуществимости на ранних стадиях, требующая определенной разработки, значительно снижает риски.

Фиксируйте темы для дальнейшего обсуждения. На семинаре всплывает масса случайных, но важных сведений: атрибуты качества, бизнес-правила, идеи по разработке пользовательского интерфейса, ограничения и т.п. Запишите их на плакатах простейшим способом, — так вы не потеряете их и продемонстрируете уважение участнику, высказавшему их. Не отвлекайтесь на обсуждение деталей, не относящихся к теме дискуссии, если только они не окажутся важными, например бизнес-правилом, которое ограничивает варианты использования.

Ограничивайте некоторые дискуссии по времени. Организатор семинара имеет право ограничить время обсуждения каждой темы, скажем, первоначально — 30 минут на каждый вариант использования. Возможно, некоторые дискуссии придется завершить позднее, но жесткие временные рамки помогают не тратить времени больше, чем запланировано на первые обсуждения темы, в результате чего может не хватить времени для обсуждения остальных тем,

Не увеличивайте размер команды и тщательно отбирайте участников. Небольшие группы работают намного быстрее. Семинары, число активных участников которых превышает пять или шесть человек, могут забуксовать, вылиться в отвлеченные разговоры и даже ссоры. Попробуйте одновременного проводить несколько семинаров — это позволит исследовать требования различных классов пользователей. В обсуждении должны участвовать сторонник продукта и другие представители пользователей, возможно, эксперт в данной предметной области, аналитик требований и разработчик. Допуском к участию в семинарах по сбору информации являются знания, опыт и право принимать решения.

Ловушка

Не допускайте в процессе сбора информации обсуждения посторонних тем, например подробностей дизайна.

Вовлекайте в обсуждение каждого. Иногда участники самоустраняются от обсуждения, например, расстроившись из-за того, что не представляют себе ценность системы. Возможно, их идеи не воспринимают серьезно, поскольку другим участникам их концепции кажутся неинтересными или группа не хочет отвлекаться от текущей дискуссии. Возможно, аутсайдер не уверен в себе и уступил право голоса более активным сотрудникам или главному аналитику. Организатору семинара

необходимо следить за языком телодвижений, чтобы разобраться в причинах замкнутости того или иного участника, и попытаться снова вовлечь его в работу. Ведь интуитивное представление каждого может оказаться очень ценным.

У семи нянек...

Работа семинаров по сбору информации о требованиях большим количеством участников может продвигаться слишком медленно. Так, моя коллега Дебби, занимавшаяся организацией семинара, где разбирались варианты использования Web-продукта, была расстроена именно этим. Двенадцать участников длинно обсуждали детали и не могли договориться. Однако темпы значительно выросли, когда Дебби сократила группу до шести человек, в число которых вошли аналитик, клиент, системный архитектор, разработчик и дизайнер. Объем обсуждаемой информации стал меньше, но темпы работы более чем компенсировали эту потерю. Участникам семинара следует по его окончании обменяться информацией с коллегами, не приглашенными на дискуссию, и обсудить высказанные идеи на следующей встрече.

Не следует ожидать, что ваши клиенты представят краткий, полный и хорошо организованный список своих потребностей. Аналитикам придется классифицировать мириады бит полученных данных о требованиях, чтобы их удалось задокументировать и использовать соответствующим образом. На рис. 7-1 показано девять таких категорий требований.

Некоторая информация не попадает ни в одну из этих категорий, в том числе:

- требования, не относящиеся к разработке ПО, например необходимость обучения пользователей работе с новой системой;
- ограничения проекта, например затраты или ограничения, налагаемые графиком (в отличие от ограничений, касающихся дизайна или реализации);
- предположения;
- требования, касающиеся данных, которые зачастую связаны с определенной функциональностью системы (вы сохраняете данные на компьютере только для того, чтобы позднее извлечь их);
- дополнительная информация хронологического или описательного характера, а также относящаяся к управлению контекстом.

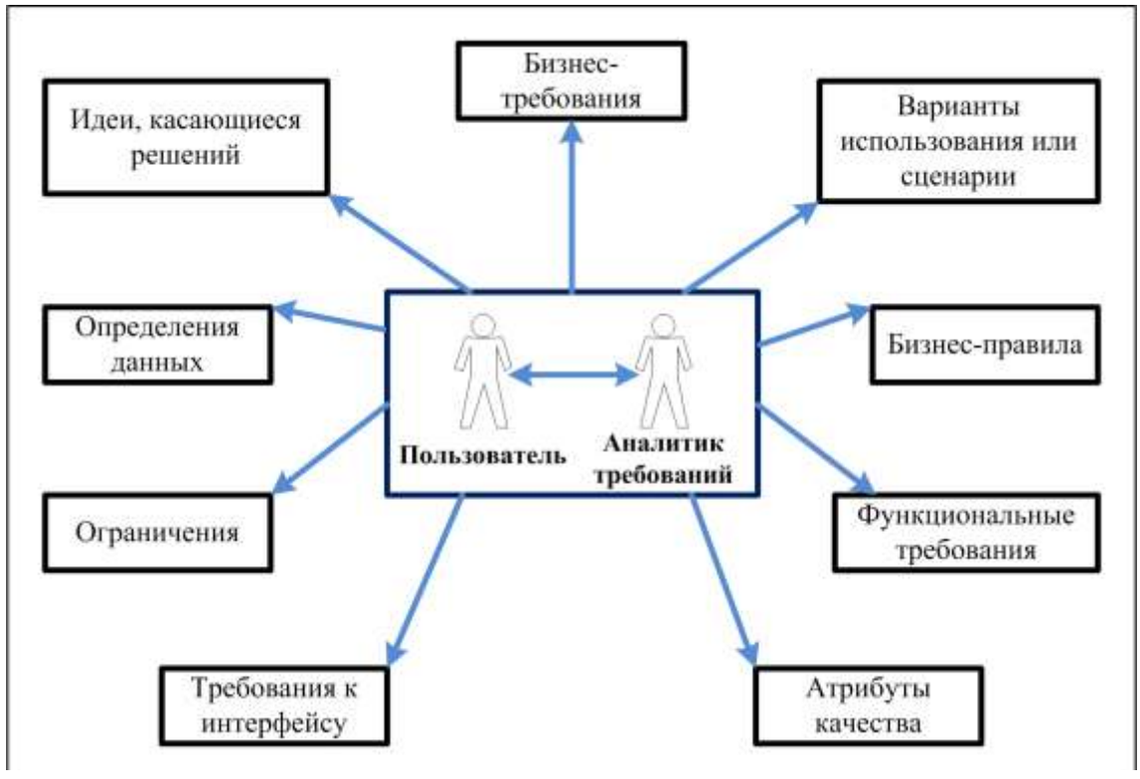


Рис. 7-1. Классификация требований клиента

Далее в этом разделе я привожу некоторые типичные для дискуссий фразы, по которым вам будет легче выполнить классификацию данных.

Бизнес-требования. Вся информация, описывающая финансовые, рыночные или другие отношения коммерческого характера, которые клиенты или разработчики собираются получить от использования продукта, относится к бизнес-требованиям. Обращайте внимание в разговоре на такие высказывания о преимуществах, которые покупатели или пользователи ПО хотят получить:

- «Увеличить рыночную долю на X%»;
- «Сэкономить Y$ в год за счет сокращения использования электроэнергии, которая сейчас расходуется неэффективно»;
- «Сэкономить Z$ за счет сокращения расходов на поддержание устаревшей системы W».

Варианты использования или сценарии. Основные утверждения пользователей о преследуемых ими целях или задачах бизнеса (коммерческих задачах) называются вариантами использования; единственный задокументированный вариант называется сценарием. Работайте с пользователями, чтобы свести сценарии в более общие варианты использования. Часто удается выявить варианты использования, попросив клиентов описать их рабочий процесс. Другой способ — выяснить у заказчиков, какие цели они ставят перед собой, усаживаясь за работу. Сотрудник, который скажет: «Мне нужно сделать то-то и то-то», вероятнее всего опишет конкретный вариант использования, например:

- «Мне нужно напечатать почтовую этикетку для пакета»;
- «Мне нужно управлять очередью образцов химических веществ отобранных для анализа»;
- «Мне необходимо откалибровать контроллер насоса».

Бизнес-правила. Если клиент заявляет, что только определенные классы пользователей могут выполнять определенные действия при определенных условиях, он, возможно, описывает бизнес-правило. В случае с Chemical Tracking System такое бизнес-правило может звучать следующим образом: «Химик может заказать вещество из списка опасных химикатов уровня 1 только при условии, что в настоящее время действует его допуск на работу с опасными соединениями». Вы можете сформулировать некие функциональные требования к ПО, чтобы определить конкретное правило, например запись в БД о допуске сотрудника должна быть доступной Chemical Tracking

System. Как уже говорилось, бизнес-правила и бизнес-требования — не одно и то же. Вот несколько фраз, по которым вы можете догадаться, что пользователь описывает именно бизнес-правила:

- «Должны совпадать с таким-то постановлением или корпоративной политикой»;
- «Должны соответствовать некоему стандарту»;
- «Если выполняется такое-то условие, то происходит то-то и то-то»;
- «Расчеты производятся по следующей формуле».

> ***Дополнительная информация***
> Развернутые примеры бизнес-правил в главе 9.

Функциональные требования. Функциональные требования описывают ожидаемое поведение системы при определенных условиях и действия, которые система позволит выполнять пользователям. Функциональные требования проистекают из системных требований, пользовательских требований, бизнес-правил и других источников, составляющих основу спецификации требований к программному обеспечению. Ниже перечислены примеры функциональных требований, которые вы можете услышать от пользователей:

- «Если давление превышает 40,0 фунтов на квадратный дюйм, должен загореться предупредительный световой сигнал»;
- «У пользователя должна быть возможность сортировать список проекта в прямом и обратном алфавитном порядке»;
- «Когда кто-либо представляет на рассмотрение новую идею, система отправляет сообщение по электронной почте Idea Coordinator».

Эти высказывания демонстрируют, как пользователи обычно представляют функциональные требования, однако их нельзя записать непосредственно в спецификацию требований к программному обеспечению. В первом случае следует заменить «должен загореться» на «загорится», чтобы стало ясно, что включение предупредительного светового сигнала крайне важно. Второй пример показывает требования пользователя, а не требования к системе. Требование к системе — предоставить пользователю возможности выполнить сортировку.

> **Дополнительная информация**
> Советы тем, кто хочет написать хорошие функциональные требования — в главе 10.

Атрибуты качества. Утверждения, насколько хорошо система соответствует определенному режиму работы или позволяет пользователям выполнять конкретные действия, называются атрибутами качества. Обращайте внимание на слова, которыми пользователи описывают желаемые характеристики системы: быстрая, легкая, интуитивно понятная, удобная для пользователя, устойчивая к сбоям, надежная и эффективная. Вам придется поработать с пользователями, чтобы понять, что именно они имеют в виду под этими неясными и субъективными терминами, и зафиксировать четкие, поддающиеся проверке атрибуты качества, как описано в главе 12.

Требования к внешнему интерфейсу. Требования в этом классе описывают связь вашей системы с остальным миром. В спецификацию требований к программному обеспечению необходимо включить разделы, касающиеся взаимодействия с пользователями, оборудованием и другими системами ПО. Вот примеры фраз, указывающих, что клиент описывает внешние требования к интерфейсу:

- «Должны распознавать сигналы от определенного устройства»;
- «Должна отправлять сообщение такой-то системе»;
- «Должны быть в состоянии читать (или записывать) файлы в определенном формате»;
- «Должна контролировать такой-то элемент оборудования»;
- «Пользовательский интерфейс должен соответствовать определенному стандарту».

Ограничения. Ограничения, касающиеся дизайна и реализации, официально ограничивают возможности, доступные разработчику. Для устройств со встроенным ПО часто необходимо учитывать физические ограничения, такие, как размер, вес и сквозные соединения в плате. Записывайте разумное обоснование для каждого ограничения, чтобы все участники проекта знали его источник и понимали его обоснованность. Действительно ли ограничением является требование разместить устройство в определенном пространстве? Или это желание, например, просто получить портативный компьютер с минимальным весом?

Введение излишних ограничений препятствует выработке лучшего решения. Из-за ограничений также не всегда удается использовать стандартные компоненты, доступные в продаже. Требование применять определенную технологию также не всегда достижимо — зачастую технология устарела или недоступна в результате прогресса. Однако некоторые ограничения могут оказаться полезными при достижении целей, связанных с атрибутами качества. Например, удается улучшить переносимость ПО путем использования только стандартных команд языка программирования и запрета на использование расширений, присущих только определенным производителям. Вот какие ограничения может выдвинуть клиент:

- «Размер файлов, представленных в электронном виде, не должен превышать 10 Мб»;
- «В браузере для всех безопасных транзакций следует использовать 128-битную кодировку»;
- «В БД необходимо применять механизм периода выполнения Framalam».

А это примеры других фраз, указывающих на то, что клиент описывает ограничения проектирования или реализации:

- «Должна быть написана на определенном языке программирования»;
- «Не может требовать более такого-то объема памяти»;
- «Должна функционировать согласованно (или быть совместима) с такой-то системой»;
- «Должна использовать определенный элемент управления пользовательского интерфейса».

Как и в случае с функциональными требованиями, задача аналитика шире, чем просто запись высказываний пользователя в спецификацию требований к программному ПО. Неудачные слова, такие, как ***согласованно и последовательно,*** необходимо уточнять, а в ограничения следует формулировать достаточно ясно, чтобы разработчики смогли принять их как руководство к действию. Поинтересуйтесь причинами ограничений, проверьте, насколько они аргументированы, и задокументируйте обоснование для включения этого ограничения в качестве требования.

Определения данных. Каждый раз, когда клиенты описывают формат, тип данных, допустимые значения или значение по умолчанию для элемента данных или структуры бизнес-данных, они дают определения данных. Например, «Почтовый индекс в США состоит из пяти цифр, за которым следует дефис и еще четыре цифры — по умолчанию 0000». Составьте словарь данных, своего рода основной справочник, который команда сможет использовать в процессе всей разработки и поддержания продукта.

Дополнительная **информация** Подробнее о словарях данных - в главе 10

Определения данных иногда становятся источниками функциональных требований, которые пользователи не запросили напрямую. Что произойдет, когда номер заказа, состоящий из шести цифр, превысит 999999? Разработчикам необходимо знать: как поведет себя система в таких случаях. Откладывая проблемы, касающиеся данных, вы только усложняете их решение в будущем. (Помните проблему 2000 г.)

Идеи, касающиеся решений. Большинство информации, которую пользователи представляют как требования, подпадает под определение идей о решении. Те, кто описывают определенный способ взаимодействия с системой для выполнения определенного действия, представляют допустимое решение. Аналитику необходимо научиться преодолевать поверхностные формулировки и докапываться до сути идеи, которая и представляет собой требование. Например, функциональные требования, касающиеся паролей — лишь одно из возможных решений требования к безопасности,

Предположим, пользователь говорит: «Затем в раскрывающемся списке я выбираю штат, куда я

хочу отправить посылку». Фрагмент *«в раскрывающемся списке»* тянет на решение. В этом случае благоразумный аналитик задаст вопрос: «Почему в раскрывающемся списке? Если пользователь ответит: «Мне просто кажется, так удобно», то непосредственно требование можно сформулировать примерно так: «Система позволит пользователю выбрать штат, куда он хочет отправить посылку». Однако пользователь может сказать: «Я предложил раскрывающийся список, потому что мы применяем этот элемент в других областях, и я хочу согласованности. Кроме того, так пользователь не сможет ввести непроверенные данные, и я думаю, что в этом случае удастся применить уже написанный код». Это отличные причины для выбора определенного решения. Однако следует помнить, что, записывая идею о решении в требование, вы тем самым налагаете ограничение по проектированию на это самое требование. Таким образом, реализация требования становится возможной только в одном варианте. Это не обязательно плохо или неправильно, просто убедитесь, что для ограничения есть убедительная причина.

Несколько советов о том, как собирать информацию

Собрать воедино мнения нескольких дюжин пользователей трудно при отсутствии хорошо структурированной организационной схемы, такой, как вариант использования. Проблемы возникают, и если вам приходится выслушивать нескольких пользователей, и если вы имеете дело с наиболее громогласным и упрямым клиентом. Вы можете упустить из виду требования, важные для определенных классов пользователей, или включить требования, которые не отражают потребности большинства. Чтобы соблюсти баланс, следует привлечь несколько горячих сторонников продукта, обладающих полномочиями для выступления от лица соответствующих классов пользователей, причем каждого из них должны поддерживать несколько представителей этого же класса пользователей.

В процессе сбора информации иногда выясняется, что границы проекта определены неправильно — они либо слишком широкие, либо слишком узкие (Christel и Kang. 1992). В первом случае вам придется собрать дополнительные данные, чтобы сформулировать адекватные коммерческие и клиентские понятия, при этом процесс неизбежно затянется. Во втором случае высказанные пользователями важные потребности выходят за границы текущего проекта. Это означает, что определенные границы, возможно, слишком малы для получения удовлетворительных результатов. Таким образом, сбор информации иногда влечет за собой изменение образа или границ проекта.

Часто говорят, что суть требований заключается в том, что система должны делать, а то, как решение будет реализовано, относится к области проектирования. Хотя эта формулировка заманчиво кратка, по сути она примитивна. (*Действительно)* сбор информации должен быть сосредоточен на этом «***что»***. Но области анализа и проектирования разделяет размытая линия, а не четкая граница. Гипотетические «***как»*** помогают уточнить и детализировать понимание потребностей пользователей. Модели анализа, наброски того, что видно на экране, и прототипы помогают в ходе сбора информации более осязаемо выразить потребности и избежать ошибок и упущений. Воспринимайте модели и проектные решения, выработанные в процессе формулирования требований, как концептуальные предложения, упрощающие эффективное взаимодействие, а не как ограничение возможностей, доступных разработчику. Объясните пользователям, что эти средства — только для иллюстрации идей и необязательно попадут в окончательные решения,

Увлечение исследованиями иногда приводит к проблемам. Высказана прекрасная идея или предложение, но для ее оценки необходима обстоятельная проверка. Выяснение ее осуществимости или ценности следует воспринимать как самостоятельную задачу внутри проекта, характеризующуюся собственными целями и требованиями. И в этом случае неоценима роль прототипов. Если для вашего проекта необходимо провести обширные исследования, выполняйте их постепенно, небольшими, безопасными частями,

Поиск упущенных требований

Пропуск каких-либо важных данных — самый распространенный недостаток требований (Jones, 1997). Обнаружить их в процессе повторного просмотра требований очень трудно, поскольку они просто невидимы! Предлагаемые далее приемы позволяют выявить упущенные требования.

Ловушка

Остерегайте паралича аналитического процесса: не тратьте слишком много времени на выявление требований, пытаясь не упустить ни одно из них. Вы никогда не выявите их все сразу!

- Раскладывайте требования высокого уровня на простейшие составляющие — это позволит понять, чего же именно просят пользователи. Из-за неясности требований высокого уровня, предоставляющих клиентам свободу интерпретации, возможно несовпадение представлений клиента о продукте и тем, что создает разработчик. Вот некоторые неточные и расплывчатые термины, которых следует избегать: ***обеспечивать поддержку, предоставлять возможность, разрешать, обрабатывать и управлять.***
- Убедитесь, что все классы пользователей предоставили вам информацию и для каждого варианта использования определена по крайней мере одна роль.
- Трассируйте требования к системе, варианты использования, списки откликов на события и бизнес-правила в детализирующие их функциональные требования. Это позволит вам быть уверенным, что аналитик описал всю необходимую функциональность.
- Для выявления недостающих требований проверяйте пограничные значения. Предположим, в одном требовании указано: «Если стоимость заказа меньше $100, стоимость доставки будет равна $5,95", а в другом — «Если стоимость заказа превышает $100, стоимость доставки составляет 5% от общей стоимости заказа». А как быть, если стоимость заказа составляет ровно $100? Это не оговорено, значит, отсутствует соответствующее требование.
- Используйте разнообразные формы представления информации о требованиях. Трудно прочитать большой объем текста и заметить, что чего-то не хватает. Модели анализа визуально представляют требования высокого уровня абстракции — лес, а не отдельные деревья. Рассматривая модель, вы можете заметить, что от одною блока к другому должна идти стрелка; это тоже недостающее требование. Подобного рода ошибку легче заметить на рисунке, чем в длинном списке требований, который сливается перед глазами. Подробнее о моделях анализа — в главе 11.

Один из точных способов поиска недостающих требований — создать матрицу CRUD (Create, Read, Update, Delete — создание, чтение, обновление, удаление). Она позволяет соотнести действия системы с элементами данных (отдельными или их совокупностями), в результате вы получаете представление о том, где и как каждый элемент денных создается, считывается, обновляется и удаляется. Некоторые добавляют к названию матрицы букву L указывая, что элемент данных является списком (List) (Ferdinandi, 2002). В зависимости от способов анализа требований, которые вы используете, удается исследовать различные типы соответствий, в том числе:

- элементы данных и системные события (Robertson и Robertson,1999);
- элементы данных и задачи пользователей или варианты использования (Lauesen, 2002);
- классы объектов и системные события (Ferdinandi, 2002);
- классы объектов и варианты использования (Armour и Miller, 2001).

Наборы требований со сложной булевой логикой (несколько операторов «И», «ИЛИ» и «НЕ») часто оказываются неполными, Если для комбинации логических условий не определено соответствующее функциональное требование, разработчику приходится догадываться, как же должна действовать система, или искать ответ на этот вопрос. Чтобы убедиться, что вы рассмотрели все возможные ситуации, представляйте сложную логику с помощью таблиц и деревьев решений (подробнее — в главе 11).

На рис. 7-2 показана матрица CRUDL для элементов данных и вариантов использования определенной области Chemical Tracking System. Каждая ячейка указывает, как вариант использования, определенный в крайнем левом столбце, взаимодействует с каждым элементом данных. Вариант использования может **Создать - (С)** (Create), **Читать – (Ч)** (Read), **Обновить – (О)** (Update), **Удалить – (Уд)** (Delete) или **Указать в виде списка – (Ук)** (List) элемент данных. После создания матрицы посмотрите, не появилась ли одна из этих букв в какой-либо ячейке столбца. Если коммерческий объект обновлен, но до этого его не создали, откуда он взялся? Обратите внимание, что ни одна ячейка в колонке с именем «Сотрудник, разместивший заказ на химикат» не содержит **«Уд».** То есть, ни в одном из случаев использования на рис. 7-2 нельзя удалить сотрудника из списка людей, заказывавших химикаты. Интерпретировать это можно тремя способами:

Элемент данных / Вариант использования	Заказ	Химикат	Сотрудник, разместивший заказ на химикат	Каталог поставщика
Местоположение заказа	**С**	**Ч**	**Ч**	**Ч, Ук**
Изменение заказа	**О,Уд**		**Ч**	**Ч, Ук**
Управление описью химикатов		**С, О, Уд**		
Сообщение о заказе	**Ч**	**Ч, Ук**	**Ч, Ук**	
Редактировать данные колонки «Сотрудник, разместивший заказ на химикат»			**С, О, Ук**	

Рис. 7-2. Пример матрицы CRUDI-для Chemical Tracking System

- удаление сотрудника, разместившего заказ на химикат, не является ожидаемой функцией Chemical Tracking System;
- мы пропустили вариант использования, который удаляет сотрудника, разместившего заказ на химикат;
- вариант использования «Редактировать данные колонки «Сотрудник, разместивший заказ на химикат»» некорректный. Предполагается, что пользователь может удалить из списка сотрудника, разместившего заказ на химикат, но в настоящее время эта функциональность не указана в вариантах использования. Мы не знаем, какая интерпретация правильна, но CRUDL — это надежный способ для обнаружения недостающих требований.

Как понять, что сбор требований завершен

Каких-либо признаков, показывающих, что выявление требований завершено, нет. Когда люди размышляют по утрам в душе и разговаривают со своими коллегами, у них возникают идеи насчет дополнительных требований. Выявить требования полностью невозможно, однако следующие признаки подскажут вам, что источники сведений уже почти иссякли:

- пользователи уже не могут придумать каких-либо еще вариантов использования. Обычно они описывают их в порядке убывания значимости последних;
- пользователи предлагают новые варианты использования, однако вы уже вывели соответствующие функциональные требования из других вариантов использования. Эти «новые» предложения могут оказаться случаями других вариантов использования, которые вы уже рассмотрели;
- пользователи повторно описывают уже обсуждавшиеся проблемы;

- предлагаемые новые функции, пользовательские или функциональные требования выходят за рамки проекта; вновь предлагаемые требования имеют низкий приоритет;
- пользователи предлагают возможности, которые имеет смысл реализовать «когда-то позже», а не включить «в конкретный продукт, который мы сейчас обсуждаем».

Еще один способ определить, что выявление требований завершено — создать контрольный список функциональных областей, общих для всех ваших проектов. К таким областям относятся регистрация ошибок, архивация и восстановление, безопасность доступа, создание отчетов, печать, функции предварительного просмотра и конфигурирование пользовательских параметров. Периодически сравнивайте этот список с уже выявленными характеристиками системы. Если расхождений не обнаружено, выявление требований, по всей видимости, завершено. Невзирая на все ваши усилия, выявить все требования вы не сможете, так что готовьтесь вносить изменения по мере работы над проектом. Помните: ваша цель — сформулировать требования, достаточные для того, чтобы обеспечить при разработке приемлемый уровень риска.

Что теперь?

- Вспомните недостающие требования, которые были выявлены на поздних стадиях проекта. Почему они были упущены из виду в процессе сбора информации? Как вы могли бы выявить их раньше?
- Выберите часть любого задокументированного мнения клиента или раздела спецификации требований к программному обеспечению. Классифицируйте каждый элемент в этой области требований по категориям, показанным на рис. 7-1: бизнес-требования, варианты использования или сценарии, бизнес-правила, функциональные требования, атрибуты качества, внешние требования к интерфейсу, ограничения, определения данных и идеи о решениях. Если вы обнаружите элементы, которые вы неправильно классифицировали, переместите их в соответствующую категорию в документации требований.
- Перечислите все методы выявления требований, использовавшиеся вами для текущего проекта. Какие методы себя хорошо зарекомендовали, а какие — нет и почему? Определите способы выявления требований, которые, по-вашему, дадут лучший результат, и решите, как вы будете применять их в следующий раз. Подумайте, какие препятствий могут помешать вам применить эти способы на практике, и определите, как их преодолеть.

Глава 8 Как понять требования пользователей

Участники проекта Chemical Tracking System проводили свой первый семинар, посвященный требованиям, чтобы выяснить, как химики будут использовать новую систему. На него были приглашены аналитик требований Лори, которая также выполняла функции ответственного за семинар, сторонник продукта от химиков Тим, еще два представителей от химиков Сэнди и Питер, а также ведущий разработчик Элен. Лори открыла семинар, поблагодарив присутствующих за участие, и сразу приступила к делу.

"Тим, Сэнди и Питер определили приблизительно 15 вариантов использования Chemical Tracking System, — сообщила она группе. — На этих семинарах мы рассмотрим их, чтобы выяснить, какую функциональность необходимо встроить в систему. Вы присвоили высший приоритет варианту использования - «Запрос химиката», и Тим уже составил его краткое описание, поэтому с него и начнем. Тим, как ты представляешь себе запрос

химиката с помощью системы?»
«Прежде всего, — сказал Тим, — вам следует знать, что только те люди, у которых есть разрешение руководителей лабораторий, могут запрашивать химикаты».
«Ладно, это похоже на бизнес-правило, — ответила Лори. — Я начну составлять список бизнес-правил, поскольку, вероятно, будут и другие. Видимо, нам придется проверять, включен ли пользователь в утвержденный список лиц». Она также написала «пользователь идентифицирован» и «пользователю предоставлено право запрашивать химикаты» в разделе «Предварительные условия» списка, который она уже подготовила для варианта использования «Запрос химиката». «Есть еще какие-нибудь предварительные условия, которые должны быть выполнены до того, как пользователь получит возможность создать запрос?»
«Прежде чем запрашивать химикат у поставщика, я должна проверить, нет ли его на складке, — сказала Сэнди. — Когда я начну составлять запрос, текущая БД складских запасов должна быть подключена в онлайновом режиме, чтобы я не теряла времени попусту».
Лори добавила этот элемент в список предварительных условий. В течение следующих 30 минут она управляла обсуждением того, как участники семинара представляют себе процесс формирования запроса нового химиката. Ей понадобилось несколько списков, чтобы свести вместе информацию о предварительных условиях, выходных условиях, а также этапах взаимодействия пользователя и Chemical Tracking System. Лори поинтересовалась, чем будет отличаться вариант использования, когда пользователь запрашивает химикат у продавца, от того, когда он запрашивается на складе. А кроме того, какие проблемы при этом могут возникнуть и как система должна обрабатывать каждую ошибку? По истечении получаса удалось выработать четкое представление о том, как пользователь будет запрашивать химикат. Участники семинара были готовы обсуждать следующий вариант использования.

Люди используют системы ПО для выполнения необходимых задач, и в индустрии ПО наблюдается устойчивая тенденция к разработке ПО с повышенной практичностью (Constantine и Lockwood, 1999; Nielsent, 2000). При этом существует обязательное предварительное условие — необходимо знать, как клиенты планируют использовать продукт.

В течение многих лет аналитики извлекали информацию о требованиях пользователей из сценариев использования (McGraw и Harbisont, 1997). ***Сценарием*** (scenario) называется описание одного варианта использования системы. Ivar Jacobson с коллегами (1992), Larry Constantine и Lucy Lockwood (1999), Alistair Cockburn (2001) и многие другие преобразовали этот подход в методику, где для сбора информации и моделирования требований применяются варианты использования. Последние прекрасно себя зарекомендовали при разработке требований для бизнес-приложений, Web-сайтов, служб, предоставляемых одной системой другой и систем, позволяющим пользователям управлять определенным оборудованием. У таких продуктов, как процессы пакетной обработки, системы для интенсивных вычислений и приложения для хранения данных, может оказаться всего несколько простых вариантов использования. Сложность их работы заключается в выполняемых вычислениях или генерировании отчетов, а не во взаимодействии пользователя и системы. Работа с требованиями, часто применяемая для систем реального времени, заключается в перечислении внешних событий, на которые система должны реагировать и соответствующих реакций системы. В этой главе описано, как с помощью вариантов использования и таблиц «событие — реакция» зафиксировать пользовательские требования.

Ловушка

Не пытайтесь включить в вариант использования каждое появившееся требование. С помощью вариантов использования удается выявить большинство, но никак не все функциональные требования.

Подход с применением варианта использования продукта

Вариант использования (use case) продукта описывает последовательность взаимодействия системы и внешнего действующего лица. ***Действующим лицом*** (actor) может быть человек, другая система ПО или аппаратное устройство, взаимодействующее с системой для достижения некой цели (Cockburn, 2001). Еще одно название действующего лица — ***роль пользователя,*** поскольку эта роль, которую члены одного или нескольких классов пользователей могут выполнять по отношению к системе (Constantine и Lockwood, 1999). Например, в варианте использования Chemical Tracking System «Запрос химиката» участвует действующее лицо — Сотрудник, разместивший заказ на химикат. Класса пользователей Chemical Tracking System с таким именем не существует. И химики, и специалисты, работающие на складе химикатов, могут запрашивать химикаты, поэтому члены любого из этих классов могут играть роль Сотрудника, разместившего заказ на химикат.

Понятие «вариант использования» пришло из мира объектно-ориентированного программирования. Тем не менее они годятся и для проектов, где применяются любые приемы разработки, поскольку пользователям безразлично, как именно создается ПО. Варианты использования лежат в основе широко применяемого Унифицированного процесса разработки ПО (Jacobson, Booch и Rumbaugh, 1999).

Варианты использования меняют традиционный подход к сбору информации; пользователей не спрашивают, как прежде, что с их точки зрения, должна делать ***система,*** а выясняют, какие задачи собирается с ее помощью решать ***пользователь.*** Цель такого подхода — описать все подобные задачи. До включения каждого варианта использования в утвержденную версию требований, заинтересованные в проекте лица проверяют, не выходит ли он за границы проекта. Теоретически в конечный набор вариантов использования должна входить вся желаемая функциональность системы. На практике же вам вряд ли удастся добиться стопроцентного результата, однако варианты использования помогут вам выполнить эту задачу полнее, чем какой-либо другой прием сбора информации, которым я когда-либо пользовался.

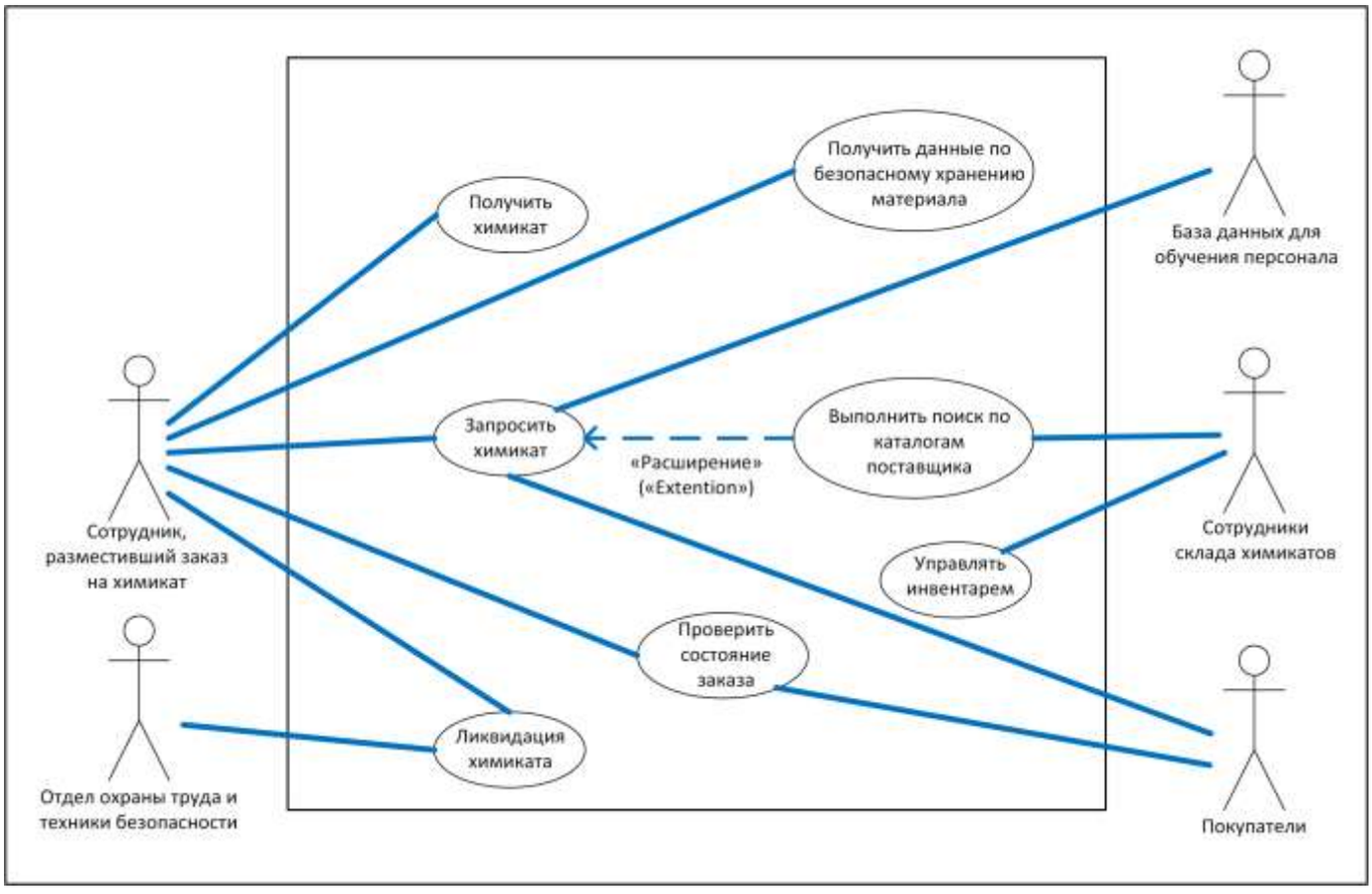


Рис. 8-1. Фрагмент диаграммы вариантов использования Chemical Tracking System

Диаграммы вариантов использования (use-case diagrams) позволяют получить отличное визуальное представление о требованиях пользователей. На рис. 8-1 показан фрагмент диаграммы варианта использования Chemical Tracking System, где применяются нотации UML (Unified Modeling Language — унифицированный язык моделирования) (Booch, Rumbaugh и Jacobson, 1999; Armour и Miller, 2001). Прямоугольник показывает границы системы. Линии соединяют каждое действующее лицо (нарисованный человечек) с вариантами использования (эллипсами), с которыми это лицо взаимодействует. Обратите внимание на сходство этой диаграммы варианта использования с контекстной диаграммой на рис. 5-3. Здесь прямоугольник отделяет некоторые внутренние элементы высокого уровня системы — варианты использования — от внешних действующих лиц. Контекстная диаграмма также отображает объекты, расположенные вне системы, но на ней не видны внутренние части системы.

Варианты использования и сценарии использования

Вариант использования (use *case)* — это отдельная, независимая деятельность, которую действующее лицо может совершить для получения определенного значимого результата. Один вариант использования может охватывать несколько схожих задач с одинаковыми целями. Следовательно, он представляет собой набор связанных между собой сценариев использования, где сценарий — это отдельный пример варианта использования. Вы можете начать разработку требований с абстрактных вариантов использования, а затем на их основе создать конкретные сценарии использования или, наоборот, перейти от некоего сценария к более широкому варианту использования. Далее в этой главе показан подробный шаблон для документирования вариантов использования. К важным элементам описания варианта использования относятся:

- уникальный идентификатор;
- имя, кратко описывающее задачи пользователи в формате «глагол + объект», например «разместить заказ»;
- краткое текстовое описание на естественном языке;
- список предварительных условий, которые должны быть удовлетворены до начала разработки варианта использования;
- выходные условия, описывающие состояние системы после успешного завершения разработки варианта использования;
- пронумерованный список действий, иллюстрирующий последовательность этапов взаимодействия лица и системы от предварительных условий до выходных условий.

Один сценарий считается ***нормальным направлением развития*** (normal course) варианта использования, его также называют основным направлением, базовым направлением, нормальным потоком, основным сценарием, главным успешным сценарием и благоприятным путем. Нормальное направление для варианта использования «Запрос химиката» — запрос химиката, который есть на складе.

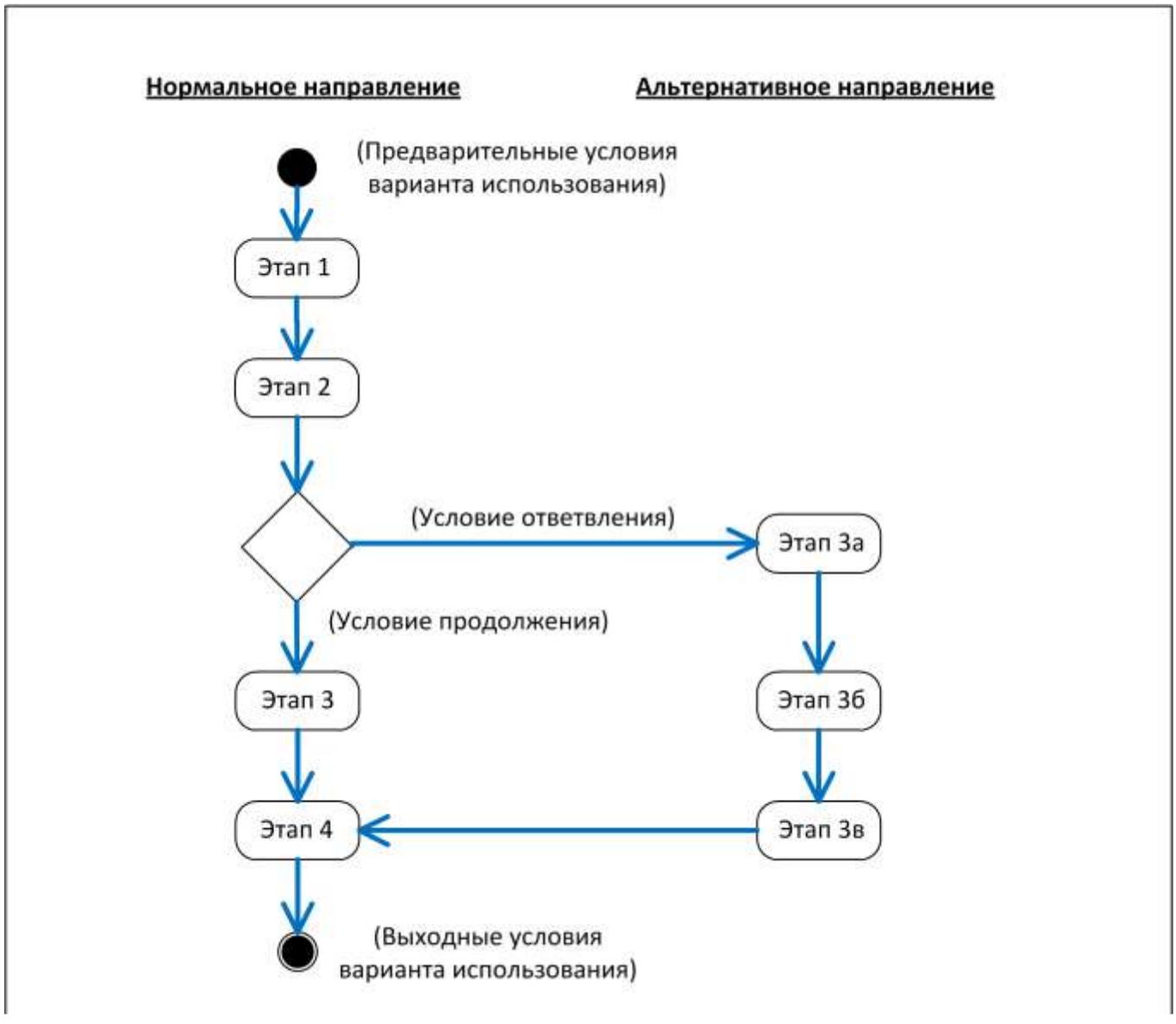


Рис. 8-2. UML-диаграмма, иллюстрирующая диалоговый поток при нормальном и альтернативном развитии варианта использования

Другие допустимые сценарии из варианта использования, называются ***альтернативными направлениями*** (alternative courses) или ***вторичными сценариями*** (secondary scenarios) (Schneider и Winters, 1998). Они также могут привести к успешному выполнению задания и удовлетворяют выходным условиям варианта использования. Однако они представляют вариации решения задачи или диалоговой последовательности, необходимой для выполнения задачи. В определенной точке принятия решений в диалоговой последовательности нормальное направление может перейти в альтернативное, а затем вернуться обратно в нормальное. Хотя большинство вариантов использования можно описать простым языком, блок-схема или UML-диаграмма позволяют визуально представить логическое развитие сложного варианта использования, как показано на рис. 8-2. Блок-схема и диаграммы взаимодействия показывают точку принятия решений и условия, при которых основное направление развития событий переходит в альтернативное.

Альтернативное направление для варианта использования «Заказать химикат» — это «Заказать химикат у поставщика». В обеих ситуациях конечная цель действующего лица одна и та же — запрос химиката, поэтому оба сценария входят в один вариант использования. Некоторые этапы альтернативного направления аналогичны этапам нормального направления, но для завершения альтернативного направления необходимо выполнить особые действия. В нашем случае пользователь

может искать необходимый химикат по каталогам поставщиков. Если альтернативное направление само по себе является автономным вариантом использования, вы можете ***расширить (extention)*** нормальное направление, включив этот отдельный вариант использования в нормальный поток (Armour и Miller, 2001). Диаграмма на рис. 8-1 иллюстрирует подобную расширенную взаимосвязь. Вариант использования «Запросить химикат» был расширен вариантом использования «Выполнить поиск по каталогам поставщика». Кроме того, сотрудники склада химикатов используют «Поиск по каталогам поставщиков» как отдельный вариант.

Иногда несколько вариантов использования имеют общие наборы этапов. Чтобы избежать повторения этих этапов в каждом варианте использования, определите отдельный варианте общей функциональностью и укажите, что он ***включен (include)*** *в* другие варианты использования как подвариант. Эта процедура аналогична вызову общей подпрограммы в компьютерной программе.

В качестве примера рассмотрим работу ПО для бухгалтерии. Возможны два варианта использования — «Оплатить счет» и «Подтвердить кредитную карту», причем в обоих, чтобы выполнить платеж, пользователь должен подписать чек. Вы можете создать отдельный вариант использования «Подписать чек», который содержит набор действий, необходимых для подписи чека. Этот вариант будет включен в обе транзакции, как показано на рис. 8-3.

Ловушка

Не затягивайте обсуждение того, когда, как и стоит ли вообще использовать взаимоотношения типа расширить и включить. Чаще всего применяется следующий прием — указать общие действия во включенном варианте использования.

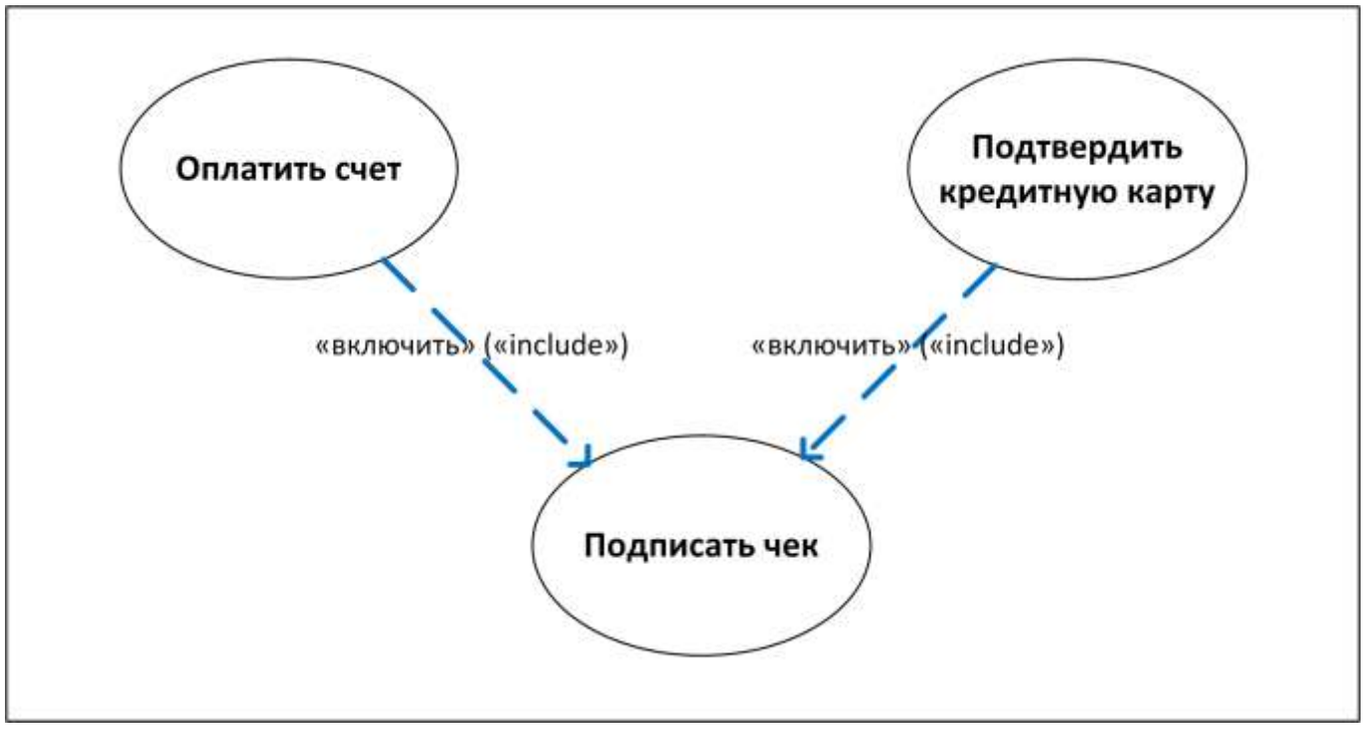


Рис. 8-3. Пример того, как вариант использования связан с бухгалтерским приложением

Условия, препятствующие успешному завершению задания, называются ***исключениями*** (exceptions). Для варианта использования «Запросить химикат» существует одно исключение — «Химиката нет в продаже». Если в процессе сбора информации вы не укажете, как обрабатывать исключение, то возможны два пути:

1. разработчики предложат лучший по их мнению способ обработки исключений;

2. при генерации пользователем неверного условия произойдет сбой системы, так как никто не предусмотрел такой ситуации.

Бьюсь об заклад, что сбои системы не входят в список требований пользователей.

Иногда исключения рассматриваются как тип альтернативного направления (Cockburn, 2001), однако эти понятия следует разделять. Не обязательно реализовывать каждое альтернативное направление, которое вы определили для варианта использования; кроме того, вы можете отложить его реализацию до следующего выпуска. Однако вы ***обязаны*** реализовать исключения, из-за которых завершение сценариев может оказаться неуспешным. Любой, кто когда-либо занимался программированием, знает, что часто именно на работу над исключениями приходится большая часть программирования. Многие недостатки конечного продукта присущи именно обработчикам исключений (или происходят из-за их отсутствия!). Указать условия исключений в ходе сбора информации по требованиям — верный способ создать устойчивые к сбоям продукты.

Для многих систем пользователь может связать последовательность вариантов использование в «макровариант использования», описывающий объемную задачу. Для коммерческого Web-сайта предлагаются, например, такие варианты использования; «Поиск по каталогу», «Добавить товар в корзину» и «Оплатить товары в корзине». Если вы можете выполнить все эти действия по отдельности, то все они считаются независимыми вариантами использования. Кроме того, вы вправе выполнить все три указанные операции подряд, назвав этот один большой вариант использования «Приобрести товар», как показано на рис. 8-4. Причем после каждого варианта использования система должна быть в таком состоянии, чтобы пользователь мог приступить к следующему варианту использования немедленно. То есть выходные условия одного варианта использования должны удовлетворять предварительным условиям следующего за ним варианта. Подобным же образом в приложении обработки транзакций, например АТМ, каждый вариант использования должен оставлять систему в состоянии готовности к следующей транзакции. Предварительные условия и выходные условия каждой транзакции варианта использования должны вставать в один ряд.

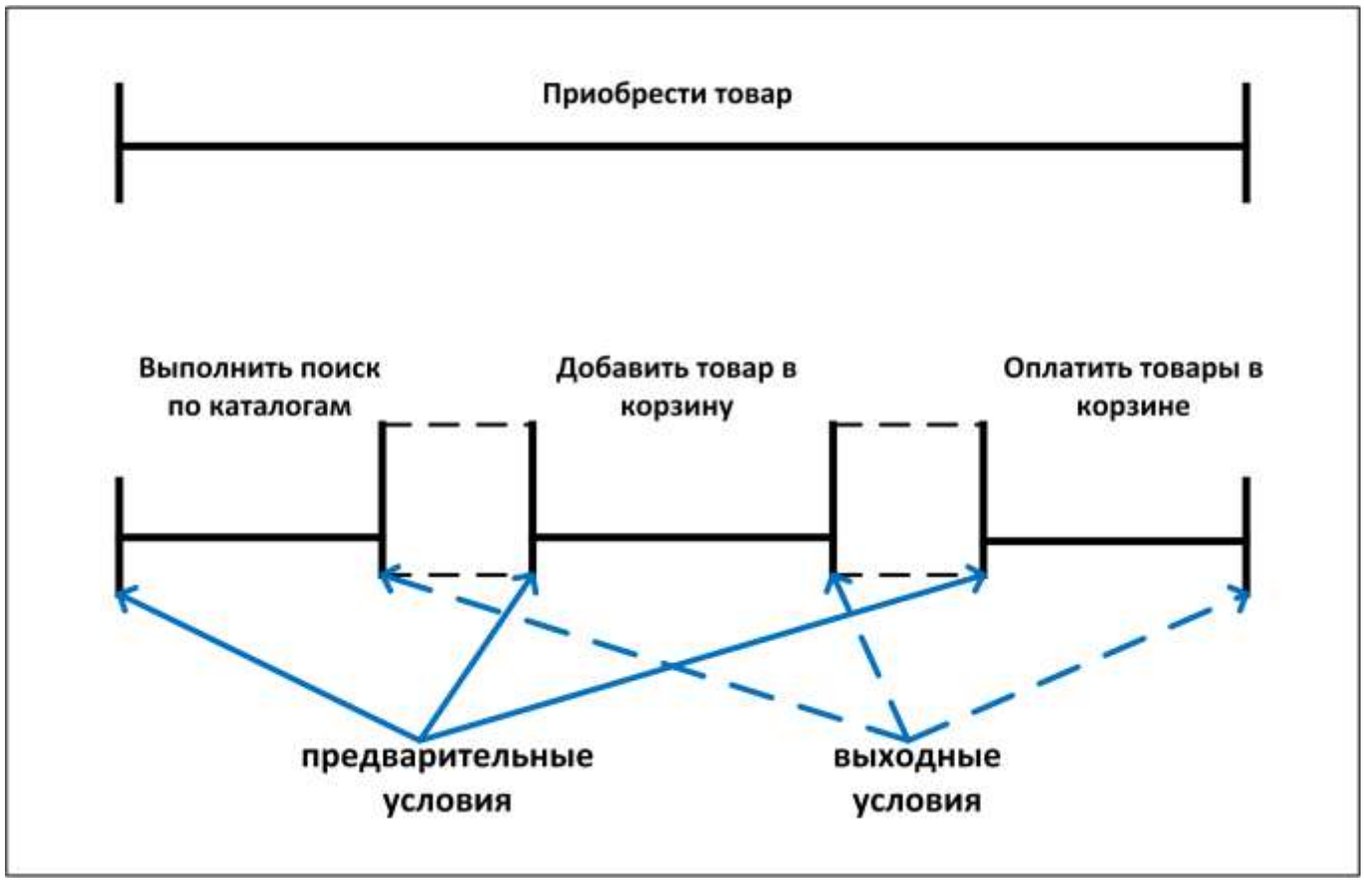


Рис. 8-4. Предварительные условия и выходные условия определяют границы отдельных вариантов использования, которые можно связать в цепочку для выполнения объемной задачи

Определение вариантов использования

Вы можете определить варианты использования несколькими способами (Ham, 1998; Larman, 1998):

- сначала определить действующие лица, а затем бизнес-процессы, в которых каждое лицо участвует;
- выразить бизнес-процессы в терминах определенных сценариев, обобщить сценарии в варианты использования и определить действующие лица для каждого варианта; определить внешние события, на которые система должна реагировать, а затем соотнести эти события с участвующими лицами и определенными вариантами использования;
- определить вероятные варианты использования на основе функциональных требований, Если какие-либо требования невозможно проследить до какого-либо варианта использования, подумайте, нужны ли они.

В Chemical Tracking System применен первый подход. Аналитик провел несколько семинаров по вариантам использования, примерно два раза в неделю, по два-три часа каждый. Члены различных классов пользователей участвовали в параллельных семинарах, так как лишь несколько вариантов использования были общими для нескольких классов пользователей. На каждом семинаре присутствовали сторонник продукта от класса пользователей, другие выбранные представители пользователей, а также разработчик. Участие в семинарах позволило разработчикам еще на ранней стадии получить представление о создаваемом продукте. Кроме того, они возвращали собравшихся к реальности, когда те предлагали невыполнимые требования.

До начала семинаров аналитики попросили пользователей продумать, какие задачи те собираются выполнять с помощью новой системы. Каждая такая задача становилась потенциальным вариантом использования. Несколько предложенных вариантов, как выяснилось, выходят за границы проекта, поэтому далее они не обсуждались. По мере изучения вариантов использования стало ясно, что некоторые варианты связаны между собой сценариями, которые можно объединить в один абстрактный вариант использования. Также удалось выявить дополнительные варианты использования, не входившие в первоначальный набор.

.Некоторые участники предложили варианты использования, не сформулированные как задачи, например «Данные по безопасному хранению материала» Название варианта использования должно указывать на решаемую задачу, поэтому в названии необходим глагол. Что клиент хочет: просмотреть, напечатать, заказать, отправить по электронной почте, исправить, удалить или создать данные по безопасному хранению материала? Иногда предложенный вариант использования — это всего лишь одно действие, которое выполняется как часть варианта использования, например «сканировать штрих-код». Аналитик должен выяснить, какую цель подразумевал пользователь, когда предлагал это. Он может задать такой вопрос: «Сканируя штрих-код на контейнере с химикатами, что вы пытаетесь сделать?» Предположим, в ответ он услышит: «Это необходимо для того, чтобы я мог отправить этот химикат в свою лабораторию». Следовательно, настоящий вариант использования выглядит как-то так — «Отправить химикат в лабораторию». Сканирование штрих-кода — только один из этапов взаимодействия действующего лица и системы, передающей химикат в лабораторию.

Как правило, пользователи сначала определяют самые важные варианты использования, поэтому порядок предлагаемых тем позволит получить представление о приоритетах. Другой способ расстановки приоритетов — кратко описать каждый потенциальный вариант использования в том виде, как он был предложен. Расставьте приоритеты и предварительно распределите их по выпускам продукта. В первую очередь укажите детали для вариантов использования с наивысшим приоритетом, чтобы разработчики смогли приступить к их реализации как можно скорее.

Документирование вариантов использования

На этой стадии обсуждения участники должны обдумать важнейшие варианты использования. Constantine и Lockwood (1999) дают следующее определение ***важнейших (абстрактных) вариантов использования*** (essential use cases): «... упрощенное, обобщенное, абстрактное, не зависящее от технологии и реализации описание одной задачи или взаимодействия... в котором воплощена цель или намерения, лежащие в основе взаимодействия». То есть следует сосредоточиться на задаче,

которую пользователю необходимо выполнить, и возможностях системы для выполнения этой задачи. Важнейшие варианты использования характеризуются более высоким уровнем абстракции, чем ***конкретные варианты использования*** (concrete use cases), которые описывают определенные действия, предпринимаемые пользователем для взаимодействия с системой. Чтобы проиллюстрировать различие, рассмотрим два способа начала реализации пользователем варианта использования «Запросить химикат».

Конкретный вариант использования. Введите номер ID химиката

Важнейший (абстрактный) вариант использования. Укажите необходимый химикат.

Формулировка на важнейшем (абстрактном) уровне позволяет пользователю выполнить задачу многими способами: ввести номер ID химиката, импортировать химическую структуру из файла, нарисовать структуру на экране с помощью мыши, выбрать химикат из списка и многое другое. Слишком быстрое углубление в детали определенного взаимодействия ограничивает широту мышления участников семинара. Независимость важнейших (абстрактных) вариантов использования от реализации, делает их более пригодными для повторных применений, чем конкретные варианты использования.

Участники семинара, посвященного Chemical Tracking System, начинали каждое обсуждение с определения действующего лица, которое получит преимущество отданного варианта использования и краткого описания этого варианта. Затем они указывали предварительные условия и выходные условия, ограничивающие вариант использования, а также все этапы внутри этих границ. Выяснив частоту использования, вы на ранних стадиях получите представление о необходимости и важности требования. Далее аналитики спрашивали участников, как те себе представляют взаимодействие с системой для выполнения задачи, Установленная последовательность действий лиц и реакции системы определялась, как нормальное направление. Нумерация этапов последовательности окончательно проясняла ситуацию. Хотя у каждого участника было свое представление об интерфейсе и специальных механизмах взаимодействия, группа смогла выработать общее понимание важнейших этапов диалога системы и пользователя.

Придерживаясь границ

Изучая вариант использования из восьми этапов, я понял, что выходные условий удовлетворены уже после пятого этапа. Следовательно, этапы 6, 7 и 8 были излишними, так как выходили за границы варианта использования. Точно так же предварительные условия варианта использования должны быть удовлетворены до начала первого этапа. Изучая описание варианта использования, убедитесь, что предварительные и выходные условия ограничивают его соответствующим образом.

Аналитик фиксировал действия отдельного лица и реакцию системы на отдельных клейких листочках, которые затем прикреплял на схему. Возможен и другой способ проведения семинара — в ходе обсуждения спроецировать с помощью компьютера шаблон варианта использования на большой экран и отредактировать его, хотя иногда это замедляет обсуждение.

Команда, занимающаяся сбором информации, разработала аналогичные диалоги для определения альтернативных направлений и исключений. Если пользователь говорит: «*По умолчанию это должно быть...*», значит, он описывает нормальное направление развития варианта использования. Такая фраза, как: «*Необходимо, чтобы пользователь также смог...*» предполагает обсуждение альтернативное направление. Многие условия исключений удавалось выяснить, когда аналитик задавал примерно такие вопросы: «Что должно произойти, если в текущий момент БД отключена?» или «Что, если этого химиката нет в продаже?». На семинаре также удобно обсудить ожидания пользователей, касающиеся качества продукта, например, времени реакции, доступности и надежности системы, ограничений дизайна пользовательского интерфейса и требований по безопасности.

После того как участники семинара описали каждый вариант использования, и никто не

предлагал уже никаких дополнительных вариантов, исключений или специальных требований, приступали к следующему варианту использования. Участники не пытались рассмотреть все варианты использования за одно марафонское обсуждение или точно определить все детали каждого варианта использования. Они запланировали изучение вариантов использования по возрастающей и их последующее многократное рассмотрение и уточнение.

На рис. 8-5 показана последовательность этапов разработки вариантов использования. После семинара аналитик детально фиксировав каждый вариант использования, как на рис. 8-6. Существует два способа представить этапы взаимодействия пользователя и системы, которые и составляют основу варианта использования. На рис. 8-6 диалог представлен в виде пронумерованного списка этапов с указанием того, какой элемент (система или определенное действующее лицо) выполняет каждый шаг. Так же описываются и альтернативные направления и исключения, для которых показан этап нормального направления развития, на котором появляется ответвление, или этап, где возможно исключение. Существует еще один прием — описать процесс средствами таблицы из двух столбцов, как показано на рис. 8-7 (Wirfs-Brock, 1993). Действия лица показаны в левом столбце, а реакция системы — в правом. Цифры указывают на последовательность этапов диалога. Эта схема отлично работает, когда с системой взаимодействует только один пользователь. Чтобы сделать эту таблицу более понятной, вы можете записать каждое действие или реакцию системы в. отдельной строке, чтобы альтернативная последовательность стала очевидной, как показано на рис, 8-8.

Часто варианты использования содержат дополнительную информацию или требования, которые не годятся ни для одного из разделов шаблона. Для таких данных предусмотрен раздел «Специальные требования»: сюда записывают соответствующие атрибуты качества, требования к производительности и др. Также зафиксируйте любую информацию, которая может быть неочевидна пользователям, например необходимое для завершения варианта использования неявное взаимодействие одной системы с другой.

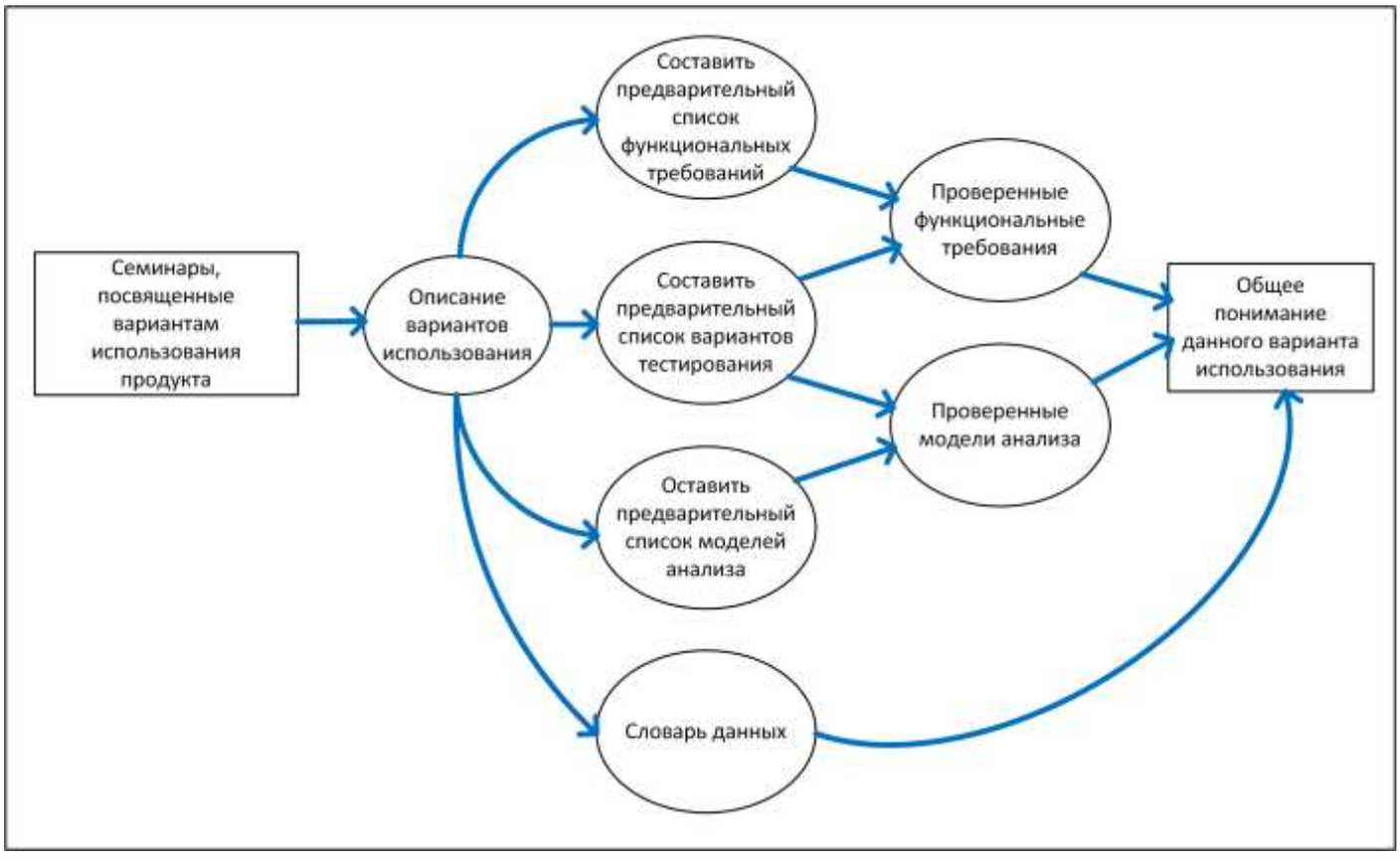


Рис. 8-5. Сбор информации для варианта использования

Идентификатор варианта использования:	Вариант использования-1	**Название варианта использования:**	Запросить химикат
Автор:	Тим	**Автор последнего обновления:**	Дженис
Дата создания:	04.12.02	**Дата последнего обновления:**	27.12.02

Действующее лицо:	Сотрудник, разместивший заказ на химикат
Описание:	Сотрудник, разместивший заказ на химикат, указывает в запросе необходимый химикат, вводя его название или номер идентификатора химиката или импортируя его структуру из соответствующего графического средства. Система выполняет запрос, предлагая новый или использованный контейнер с химикатом со склада или позволяя создать запрос на заказ у стороннего поставщика.
Предварительные условия:	1. Личность пользователя аутентифицирована. 2. Пользователь имеет право запрашивать химикаты. 3. База данных ПО запасам химикатов в данный момент подключена.
Выходные условия:	1. Запрос сохраняется в Chemical Tracking System. 2. Запрос был отправлен по электронной почте на склад химикатов или поставщику.
Нормальное направление развития варианта использования:	**1.0 «Запросить химикат со склада»** 1. Сотрудник указывает требуемый химикат. 2. Система подтверждает, что такой химикат доступен, 3. Система перечисляет контейнеры с необходимым химикатом, имеющиеся на складе. 4. Сотрудник может просмотреть историю любого контейнера. 5. Сотрудник выбирает определенный контейнер или просит отправить запрос поставщику (альтернативное направление 1.1). 6. Сотрудник вводит остальную информацию, чтобы завершить запрос. 7. Система сохраняет запрос и отправляет его по электронной почте на склад химикатов.
Альтернативное направление развития варианта использования:	**1.1 «Запросить химикат у поставщика»** (ответвление после этапа 5) 1. Сотрудник ищет химикат по каталогам поставщика. 2. Система отображает список поставщиков, где также указаны размеры, класс и цена контейнеров. 3. Сотрудник выбирает поставщика, размер, класс и количество контейнеров. 4. Сотрудник вводит остальную информацию, чтобы завершить запрос. 5. Система сохраняет запрос и отправляет его по электронной почте поставщику.

Исключения:	**1.0.И. 1 — «Химикат недоступен»** (на этапе 2). 1. Система отображает сообщение «Химикат не существует». 2. Система спрашивает сотрудника, хочет ли он запросить другой химикат или выйти из программы. 3а. Сотрудник просит запросить другой химикат. 4а. Система заново начинает нормальное направление развития варианта использования. 3б. Сотрудник решает выйти из системы. 4б. Система завершает вариант использования. **1.0.И.2. «Химиката нет в продаже»** (на этапе 5). 1. Система отображает сообщение «Нет поставщика этого химиката». 2. Система спрашивает сотрудника, хочет ли он запросить другой химикат или выйти из программы. 3а. Сотрудник запрашивает другой химикат. 4а. Система заново начинает нормальное направление развития варианта использования, 3б. Сотрудник решает выйти из системы. 4б. Система завершает вариант использования.
Включение:	**Вариант использования-22 — «Просмотреть хронологию контейнера»**
Приоритет:	Высокий
Частота использования:	Используется примерно пять раз в неделю каждым химиком, 100 раз в неделю каждым работником склада.
Бизнес-правила:	**Бизнес-правнпо-28 — «Только сотрудник, получивший разрешение заведующего лаборатории, может запрашивать химикаты».**
Специальные требования:	1.Система должна импортировать химические структуры в стандартной зашифрованной форме из любых средств, поддерживающих рисование химических структур.
Предположения:	1.Импортированные химические структуры должны быть достоверными,
Замечания и вопросы:	1. Тим выяснит, нужно ли одобрение руководства для запроса химиката, относящегося к уровню 1 списка опасных химикатов. Дата выполнения: 04.01.03.

Рис. 8-6. Описание варианта использования •«Запросить химикат».

Действия лица	Реакция системы

1.Укажите необходимый химикат, 4.Если нужно, запросите историю любого контейнера. 5.Выберите определенный контейнер (выполнено) или попросите отправить заказ поставщику (альтернативное направление 1.1)	2. Убедится, что такой химикат существует. 3. Отобразит список контейнеров с необходимым химикатом, в данный момент числящихся на складе.

Рис. 8-7. Описание этапов варианта использования в двухстолбцовом формате

Действия лица	**Реакция системы**
1.Укажите необходимый химикат.	
	2.Убедится, что такой химикат существует. 3.Отобразит список контейнеров с необходимым химикатом, в данный момент числящихся на складе.
4.Если нужно, запросите историю любого контейнера, 5.Выберите определенный контейнер (выполнено) или попросите отправить заказ поставщику (альтернативное направление 1.1).	

Рис. 8-8. Альтернативный макет для описания этапов варианта использования в двухстолбцовом формате

Вам не всегда необходимо полное описание варианта использования. Alistair Cockburn (2001) Дописывает ***рабочий*** (casual) и ***полный*** (fully dressed) шаблоны вариантов использования (последний показан на рис. 8-6). Рабочий вариант использования — это просто текстовое изложение целей пользователя и взаимодействий с системой; что-то вроде раздела «Описание» на рис. 8-6. Рассказы пользователей, которые рассматриваются как требования в экстремальном программировании, представляют собой, по существу, рабочие версии варианта использования, как правило, написанные на учетных карточках (Jeffries, Anderson и Hendrickson, 2001). Полные описания вариантов использования необходимы, если:

- в работе над проектом представители пользователей действуют не в тесной связи с разработчиками;
- приложение сложное, и высок риск сбоев системы;
- описания вариантов использования — это самый низкий уровень детализации требований, предназначенный для разработчиков;
- вы планируете разработать подробные варианты тестирования на основании пользовательских требований;
- для совместной работы территориально удаленных друг от друга команд необходимо детальные общие данные.

Ловушка

Не пытайтесь догматически определять количество деталей, которые необходимо включить в вариант использования; помните, что ваша задача - достаточно хорошо разобраться, для чего система нужна пользователям, чтобы разработчики могли заняться приложением, не опасаясь необходимости будущих переделок.

На рис. 8-5 показано, что после каждого семинара аналитики Chemical Tracking System выявляли функциональные требования к ПО на основе описаний вариантов использования (подробнее об этих темах — в следующем разделе главы). Они также создали модели анализа для некоторых сложных вариантов использования, например диаграмму перехода состояний, показывающую все возможные состояния запроса химиката и все допустимые изменения состояний.

Спустя день или два после семинара аналитик раздал описания вариантов использования и функциональных требования участникам, чтобы те просмотрели их до начала следующего семинара. Эти неофициальные просмотры выявили множество ошибок: ранее не выявленные альтернативные направления, новые исключения, некорректные функциональные требования и пропущенные этапы диалога. Оставьте между семинарами хотя бы один день. Умственное расслабление, которое наступает, когда минует день или два после мозгового штурма, позволяет людям взглянуть на проделанную работу с новой точки зрения. Один аналитик, проводивший семинары каждый день, заметил, что участникам стало трудно находить ошибки в просматриваемых документах, потому что информация была еще слишком свежа в памяти. Они прокручивали в уме дискуссии, которые только что завершились, и не замечали ошибок.

Дополнительная информация

В главе 11 описано несколько моделей анализа для Chemical Tracking System.

Ловушка

Не ждите окончания сбора информации по требованиям, чтобы просить пользователей заняться просмотром собранного материала.

На ранних стадиях разработки Chemical Tracking System руководитель тестирования создал концептуальные варианты тестирования, не зависящие от специфики реализации, на основе вариантов использования (Collard, 1999). Эти варианты тестирования помогли прийти команде к общему четкому пониманию того, как система должна функционировать при реализации определенных сценариев использования. Эти варианты тестирования позволили аналитикам проверить, действительно ли выявлены все функциональные требования, необходимые для выполнения пользователями каждого варианта использования. На заключительном семинаре, участники вместе провели критический анализ вариантов тестирования, чтобы убедиться, что одинаково понимают, как должны работать варианты использования. Как и при любом контроле качества, они нашли ошибки и в требованиях, и в вариантах тестирования.

Дополнительная информация

В главе **15** детально обсуждается, как составить варианты тестирования на основании требований.

Варианты использования продукта и функциональные требования

Разработчики ПО не реализуют бизнес-требования или варианты использования. Они реализуют функциональные требования, определенные фрагменты поведения системы, позволяющие клиентам выполнять варианты использования и выполнять свои задачи. Варианты использования описывают поведение системы с точки зрения действующего лица, при этом упускается множество деталей. Чтобы разработать и реализовать систему должным образом разработчику необходимо ознакомиться с множеством точек зрения.

Некоторые практики считают, что варианты использования это и есть функциональные требования. Однако мне случалось видеть, как возникали проблемы из-за того, что варианты использования просто передавались разработчикам для реализации. Варианты использования описывают точку зрения пользователя, его взгляд на внешнее, видимое поведение системы. В них не содержится всей информации, которая необходима разработчику для написания ПО, Так, человеку, использующему банковский автомат, не важно, какие неочевидные действия этот автомат должен выполнить, например взаимодействовать с компьютером банка. Эти детали невидимы для пользователя, однако разработчик должен о них знать. Естественно, в описания вариантов использования могут входить такие детали подобных неочевидных процессов, однако, как правило в ходе дискуссий с пользователями такие вопросы не поднимаются. Даже у разработчиков, которым передали полные варианты использования, часто возникает много вопросов. Чтобы уменьшить эту неопределенность, я рекомендую аналитикам требований ясно расписывать детали функциональных требований, необходимых для реализации каждого варианта использования (Arlow, 1998).

Многие функциональные требования не записаны в последовательности диалогов действующего лица и системы. Некоторые из них очевидны, например: «Система назначит уникальный порядковый номер каждому запросу» Нет смысла повторять эти детали в спецификации требований к ПО, если они совершенно ясно указаны в варианте использования. Другие функциональные требования не входят в описание варианта использования. Их выявляет аналитик на основании общего представления о варианте использования и операционной среды системы. Преобразование требований, как их видит пользователь, в форму, полезную разработчикам, — задача аналитика, один из многих способов, позволяющих ему обогатить проект.

В Chemical Tracking System варианты использования в основном применялись в качестве механизма для определения необходимых функциональных требований. Аналитики составляли только рабочие описания наименее сложных вариантов использования. Затем они выявляли все функциональные требования, которые после реализации позволяли выполнять все варианты использования, в том числе альтернативные направления и обработчики исключений. И далее документировали эти функциональные требования в спецификации требований к ПО, сформированной с учетом особенностей продукта.

Задокументировать функциональные требования, связанные с вариантом использованием, можно несколькими способами. Выбор способа зависит от того, как ваша команда будет выполнять проектирование, сборку и тестирование — на основе документации о вариантах использования, спецификации требований к ПО или применяя оба этих документа. Ни один из этих методов нельзя назвать идеальным, поэтому выберите тот, что лучше позволит вам документировать и управлять требования к ПО вашего проекта.

Только варианты использования

Вы можете включить функциональные требования непосредственно в описание каждого варианта использования. Независимо от этого вам понадобится отдельная дополнительная спецификация для фиксирования нефункциональных и всех функциональных требований, которые не связаны с конкретными вариантами использования. Несколько вариантов использования могут нуждаться в одном и том же функциональном требовании, Если для пяти вариантов использования потребуется аутентификация пользователя, вам вряд ли захочется писать пять разных блоков кода. Вместо их повторения, установите перекрестные ссылки на функциональные требования, присутствующие в нескольких вариантах использования. Иногда вариант использования ***включает*** взаимосвязи, о которых говорилось ранее в этой главе и которые являются решением проблемы.

Варианты использования и спецификация требований к ПО

Другая возможность — в довольно простой форме описать варианты использования и

задокументировать функциональные требования, выявленные из каждого варианта использования в спецификации требований к ПО. В этом случае вам придется установить трассируемость между вариантами использования и связанными с ними функциональными требованиями. Лучше всего управлять трассируемостью, если вы сохраните все варианты использования и функциональные требования в средстве управления требованиями.

Дополнительная информация

Подробнее о средствах управления требованиями - в главе **21.**

Только спецификация требований к ПО

Третий способ — упорядочить спецификацию требований к ПО по вариантам использования или функциям и включить последние в спецификацию. Именно его и применяли специалисты, работавшие над Chemical Tracking System. В этой схеме нет отдельных документов для вариантов использования. Вы определяете повторяющиеся функциональные требования или указываете каждое функциональное требование только один раз и ссылаетесь на него при каждом появлении его в другом варианте использования.

Преимущества способа с применением вариантов использования

Преимущества применения вариантов использования в том, что каждый вариант сосредоточен на поставленной задаче и пользователе. Пользователи будут более четко представлять, что же им даст новая система, если вы выберете способ, который фокусируется на функциях системы. При разработке нескольких Интернет-проектов представители клиентов заявили, что варианты использования позволили им более четко представить, какие возможности должны получить посетители их Web-сайтов. А аналитики и разработчики смогли разобраться и в бизнесе-процессах пользователей, и в предметной области. Тщательное изучение этапов взаимодействия лица и системы помогает еще на ранних стадиях разработки выявить неясности и неточности, а также позволяет составить варианты тестирования на основе вариантов использования.

Очень расточительно и болезненно для разработчиков писать код, которым никогда не удастся воспользоваться. Если вы будете заблаговременно определять требования и включать в них все мыслимые функции, то рискуете создать избыточные требования. Способ с применением вариантов использования позволяет выявить функциональные требования, с помощью которых пользователи будут выполнять конкретные задачи. Так вы предотвратите появление «функций-сирот», тех, что в ходе сбора информации казались весьма полезными, однако которыми никто не будет пользоваться из-за того, что они не связаны напрямую с решением рабочих задач.

Способ с применением вариантов использования облегчает расстановку приоритетов требований. Высшим приоритетом обладают те функциональные требования, которые созданы на основе вариантов использования с высшим приоритетом. Высший приоритет назначается по следующим причинам:

- варианты использования описывают один из основных бизнес-процессов, активизируемых системой;
- многие пользователи часто обращаются к ним;
- их запросил привилегированный класс пользователей;
- они предоставляют возможности, необходимые для соответствия требованиям;
- функции других систем зависят от их наличия.

Ловушка

Не тратьте много времени на обсуждение деталей вариантов использования, которые не будут реализованы в ближайшие месяцы или годы. Вероятнее всего, они изменятся еще до начала сборки.

Существуют также и преимущества технического характера. С помощью варианта

использования можно выявить некоторые важные объекты предметной области и их взаимоотношения. Разработчики, использующие объектно-ориентированные методы проектирования, могут преобразовать варианты использования в объектные модели, такие, как диаграммы классов и диаграммы последовательностей. (Однако следует помнить, что варианты использования ни в коем случае не ограничиваются разработкой объектно-ориентированных проектов). По мере того как с течением времени изменяются бизнес-процессы, задачи, реализуемые посредством определенных вариантов использования, также изменяются. Если вы проследите функциональные требования. Проектные решения, код и тесты вплоть до их истоков — пожеланий клиента — вам будет легче внести эти изменения бизнес-процессов во всю систему.

Каких ловушек следует опасаться при способе с применением вариантов использования

Как и любой другой способ разработки ПО, применение вариантов использования чревато возникновением множества проблем (Kulak и Guiney, 2000; Lilly, 2000).

- **Слишком много вариантов использования.** Если вас захлестнул вал вариантов использования, вам, возможно, не удастся записать каждый из них на соответствующем уровне абстракции. Не создавайте отдельный вариант использования для каждого возможного сценария. Лучше включите нормальное направление развития, альтернативные направления и исключения в виде сценариев в один вариант использования. Как правило, количество вариантов использования превышает количество бизнес-требований и функций, однако функциональных требований обычно бывает намного больше, чем вариантов использования.
- **Очень сложные варианты использования.** Однажды я изучал вариант использования объемом в четыре страницы, плотно заполненных описанием этапов взаимодействия, встроенной логики и условий ответвления. Документ казался весьма невразумительным. Вам не дано контролировать сложность бизнес-задач, но вы можете выбрать способ их представления в вариантах использования. Выберите один успешный способ выполнения варианта использования, с одной комбинацией значений правильных и ложных значений для различных логических решений и назовите его нормальным направлением. Другие успешные логические ответвления определите как альтернативные направления и назначьте исключения для обработки неудачных ответвлений. Альтернативных направлений может быть множество, однако каждое должно быть кратким и понятным. Чтобы не усложнять эту схему, записывайте варианты использования в терминах, основных для взаимодействий пользователя и системы, без особой детализации.
- **Включение пользовательского интерфейса в варианты использования.** Варианты использования должны описывать то, что пользователям необходимо выполнить с помощью системы, а не на то, как это будет выглядеть на экране. Сосредоточьтесь на концептуальном взаимодействии пользователя и системы, отложите работу над пользовательским интерфейсом до стадии проектирования. Например, правильная формулировка на этой стадии — «система предоставляет выбор», а не «система отображает раскрывающийся список». Не допускайте, чтобы дизайн пользовательского интерфейса диктовал направление разработки требований. Применяйте наброски экрана и карты диалогов (диаграммы архитектуры интерфейса, как описано в главе 11), чтобы в визуальной форме представить взаимодействия исполнителя и системы, а не утвердить дизайн.
- **Включения определения данных в варианты использования,** Мне приходилось видеть варианты использования, которые включали в себя определения элементов данных и структур, которыми манипулируют в варианте использования. Участникам проекта было трудно отыскать в них необходимые определения, поскольку неясно, в каком именно варианте использования оно содержится. В *таких* случаях возможен повтор определений, а значит, и их рассинхронизация, когда один экземпляр изменяется, а остальные нет. Вместо того чтобы «разбрызгивать» определения по вариантам использования, соберите их данных в словарь данных, созданный для всего проекта, как

описано в главе 10.

- **Варианты использования, которые непонятны пользователям.** Если пользователи не видят связи описания вариантов использования со своими бизнес-процессами или задачами, то вариант использования следует подкорректировать. Составляйте варианты использования с точки зрения клиентов, а не системы и попросите клиентов проверить их. Возможно, вам не нужны излишне сложнее полные варианты использования.
- **Новые бизнес-процессы.** Пользователям будет трудно придумывать новые варианты использования, если ПО создается для поддержки процесса, которого еще нет. В этом случае сбор информации по требованиям может оказаться не моделированием бизнес-процесса, а изобретением такового. Рискованно ожидать, что с помощью новой информационной системы будет создан эффективный бизнес-процесс. Выполните модернизацию бизнес-процесса, прежде чем браться за полное описание новой информационной системы.
- **Слишком частое упоминание слов расширить и включить.** Начиная работу с вариантами использования, необходимо изменить отношение аналитиков, пользователей и разработчиков к требованиям. Тонкости взаимоотношений, определяемых словами расширить и включить, могут запутать кого угодно. Лучше избегать их, пока вы не освоите способ с применением вариантов использования.

Таблицы «событие - реакция»

Есть еще один способ систематизации и документации пользовательских требований: определить внешние события, на которые система должна реагировать. ***Событием*** (event) называется какое-либо изменение или действие в среде пользователя, вызывающее реакцию системы ПО (McMenamin и Palmer, 1984; Wiley, 2000). В ***таблице «событие-реакция»*** *(*event-response table) [ее также называют ***таблицей событий*** (event table) или ***списком событий*** (event list)] перечислены все такие события и ожидаемое поведение системы, которое должно последовать, как реакция на каждое событие. Существует несколько типов системных событий, они показаны на рис. 8-9 и описаны далее:

- взаимодействие пользователя с ПО, как когда он, например, вызывает вариант использования [иногда называется ***бизнес-событием*** (business event)]. Последовательность событий и реакций соответствует этапам взаимодействия для этого варианта использования, В отличие от вариантов использования, в таблице «событие - реакция» не описываются цели пользователя при работе с системой и не приводятся причины, по которым эта последовательность событий и реакций имеет значение для пользователя;
- контрольный сигнал, чтение данных или прерывание, полученное от внешнего аппаратного устройства, например при изменении положения выключателя, изменении напряжения или перемещении мыши пользователем;
- событие инициируется в определенный момент времени (скажем, автоматический запуск экспорта данных в полночь) или когда истекает предварительно заданный период времени с момента определенного события (например, система фиксирует температуру, считываемую сенсором каждые 10 секунд).

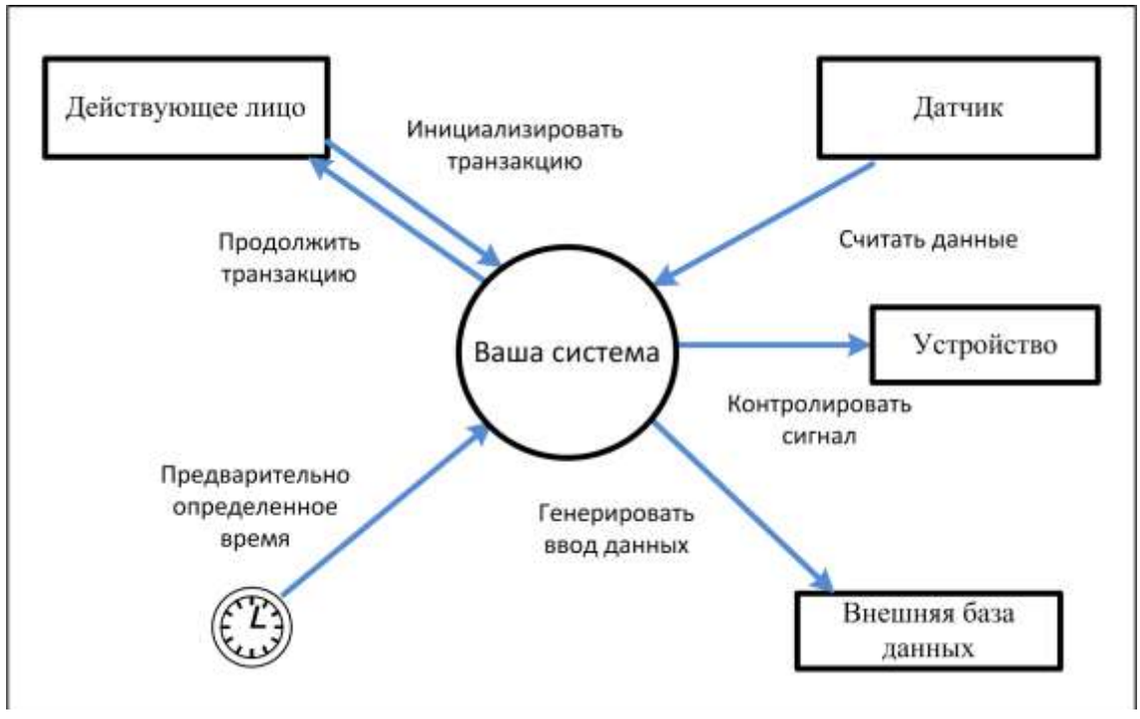


Рис. 8-9. Примеры системных событий и реакций.

Таблицы «событие - реакция» особенно хороши для управления системами, работающими в режиме реального времени. Табл. 8-1 представляет собой простую таблицу «событие — реакция», в которой частично описано поведение очистителей ветрового стекла автомобиля. Обратите внимание, что ожидаемая реакция зависит не только от события, но и от состояния, в котором находится система во время события. Например, результаты событий 4 и 5.1 в табл. 8-1 различаются из-за того, были ли включены «дворники» в момент, когда пользователь настроил управление «дворниками» на периодический режим. Реакция может просто изменить некоторую внутреннюю системную информацию (события 4 и 7.1 в таблице) или привести к результату, заметному извне (большинство других событий).

Информация в таблице «событие — реакция» фиксируется на уровне пользовательских требований. Если в таблице определены и названы все возможные комбинации событий, состояний и реакций (включая условия исключений), таблица также может служить частью функциональных требований для этого фрагмента системы. Однако аналитик должен добавить дополнительные функциональные и нефункциональные требования в спецификацию требований к ПО. Например, сколько «взмахов» в минуту делают «дворники» в быстром и медленном режиме очищения? В быстром или медленном режиме выполняется периодическая очистка? Является ли периодический режим непрерывно изменяющимся или он работает дискретно? Каковы максимальный и минимальный интервал времени при работе в периодическом режиме? Если вы закончите работу на уровне пользовательских требований, разработчику самому придется выяснять эту информацию. Помните, ваша цель - сформулировать требования настолько точно, чтобы разработчик знал, что строить.

Обратите внимание, что события перечисленные в табл. 8-1 записаны на ***важнейшем (абстрактном) уровне*** (описывается суть события), а не на уровне ***реализации*** (описываются особенности реализации). В табл. 8-1 ничего не говорится о том, как выглядит элемент управления «дворниками» или о том, как пользователь управляет ими. Дизайнер может реализовать эти требования как угодно — от традиционных элементов управления «дворниками», как в современных автомобилях, до системы распознавания голоса, реагирующей на произносимые команды: «включить дворники», «выключить дворники», «быстрая очистка», «очистить один раз» и т.д. Записывая пользовательские требования на важнейшем (абстрактном) уровне, вы избежите введения ненужных ограничений проектных решений. Однако следует фиксировать все известные ограничения по дизайну (проектированию), чтобы заставить разработчика думать.

Таблица 8-1. Таблица «событие - реакция» для автомобильных «дворников»

ID	Событие	Состояние системы	Реакция системы
1.1	установка системы управления «дворниками» в режим медленной	«дворники» отключены	установка механизма «дворников» в режим медленной очистки
1.2	установка системы управления «дворниками» в режим медленной	быстрая очистка	установка механизма «дворников» в режим медленной очистки
1.3	установка системы управления «дворниками» в режим медленной	периодическая очистка	установка механизма «дворников» в режим медленной очистки
2.1	установка системы управления «дворниками» в режим быстрой очистки	«дворники» отключены	установка механизма «дворников» в режим быстрой очистки
2.2	установка системы управления «дворниками» в режим быстрой очистки	медленная очистка	установка механизма «дворников» в режим быстрой очистки
2.3	установка системы управления «дворниками» в режим быстрой очистки	периодическая очистка	установка механизма «дворников» в режим быстрой очистки
3.1	отключение системы управления «дворниками»	быстрая очистка,	1. завершение текущего цикла очистки 2. выключение механизма «дворников»
3.2	отключение системы управления «дворниками»	медленная очистка	1. завершение текущего цикла очистки 2. выключение механизма «дворников»
3.3	отключение системы управления «дворниками»	периодическая очистка.	1. завершение текущего цикла очистки 2. выключение механизма «дворников»
4	установка системы управления «дворниками» в режим периодической очистки	«дворники» отключены	1. считывание временного интервала очистки 2, инициализация таймера очистки
5.1	установка системы управления «дворниками» в режим периодической очистки	быстрая очистка	1. считывание временного интервала очистки 2. завершение текущего цикла очистки 3. инициализация таймера очистки
5.2	установка системы управления «дворниками» в режим периодической очистки	медленная очистка	1. считывание временного интервала очистки 2. завершение текущего цикла очистки 3. инициализация таймера очистки
6	со времени предыдущей очистки прошел установленный интервал времени	периодическая очистка	выполнение одного цикла медленной очистки
7.1	изменение интервала периодической очистки	периодическая очистка	1. считывание временного интервала очистки 2. инициализация таймера очистки
7.2	изменение интервала периодической очистки	«дворника отключены	реакция отсутствует

7,3	изменение интервала периодической очистки	быстрая очистка	реакция отсутствует
7.4	изменение интервала периодической очистки	медленная очистка	реакция отсутствует
8	получение сигнала на выполнение периодической очистки	«дворники» отключены	выполнение одного цикла медленной очистки

Что теперь?

- Составьте несколько вариантов использования для вашего текущего проекта на основании шаблона на рис. 8-6. Включите в них все альтернативные направления развития и исключения. Определите функциональные требования, которые позволят пользователю успешно завершить каждый вариант использования. Проверьте, не включены ли уже все эти функциональные требования в спецификации требований к ПО.
- Перечислите все внешние события, которые могут вызвать определенное поведение системы. Создайте таблицу «событие - реакция», в которой показано состояние системы в момент воздействия каждого события и реакция системы на него.

Глава 9 Игра по правилам

«Привет, Тим, это Джекки. У меня проблема с запросом химиката при помощи системы контроля химикатов. Мой менеджер посоветовал обратиться к тебе. Он сказал, что ты был сторонником продукта и занимался требованиями к этой системе». «Да, это так, — ответил Тим.— А в чем проблема?»
«Мне нужно еще немного фосгена для красок, которыми я сейчас занимаюсь, — сказала Джекки, — но система не принимает мой запрос и сообщает, что я уже больше года не посещала занятий по работе с опасными химикатами. О чем это она? Я годами работала с фосгеном и еще более страшными химикатами — и никаких проблем. Почему же мне нельзя получить еще немного фосгена?»
«Ты, возможно, знаешь, что Contoso Pharmaceuticals требует от сотрудников ежегодно прослушивать курс по безопасной работе с химикатами, — заметил Тим. — Такова корпоративная политика; система контроля химикатов лишь проводит ее в жизнь. Я знаю, что раньше кладовщики давали тебе все, что требовалось, но теперь из-за страховки это невозможно. Извини за неудобства, но мы должны подчиняться правилам. Тебе придется пройти обучение, чтобы система выполнила твой запрос».

В любой организации действует широкий набор корпоративных политик, законов и промышленных стандартов. Такие отрасли, как банковское дело, авиация и производство медицинской техники, подчиняются постановлениям правительства. Все эти контролирующие принципы в целом называются ***бизнес-правилами*** (business rules). Зачастую их выполнение и соблюдение осуществляется средствами программных систем. В других случаях за соблюдением правил и процедур следят люди.

Большинство бизнес-правил берут начало вне контекста какой-либо конкретной программной системы. Такие правила, как корпоративная политика, обязывающая ежегодно прослушивать курс по работе с опасными химикатами, действуют, даже если приобретение и распространение химикатов осуществляется исключительно вручную. Стандартные правила отчетности существовали даже в дни зеленых наглазных повязок для сна и чернильных авторучек. Тем не менее бизнес-правила — один из основных источников функциональных требований к ПО, поскольку они определяют возможности, которыми должна обладать система для выполнения правил. Даже бизнес-требования высокого уровня могут определяться бизнес-правилами; так, например, система контроля химикатов должна генерировать отчеты о выполнении федеральных и государственных постановлений о хранении и использовании химикатов.

Не все фирмы рассматривают собственные важнейшие бизнес-правила как ценность, которой они и являются на самом деле. Если эта информация не задокументирована и не хранится должным образом, она существует только в головах сотрудников. Но последние могут по-разному понимать правила, в результате чего отдельные программные системы могут по-разному выполнять одно и то же бизнес-правило. Если вам известно, где и как каждое приложение реализует относящиеся к нему бизнес-правила, модифицировать приложения в случае изменения правил станет гораздо легче.

Ловушка

Недокументированные бизнес-правила известны только отдельным специалистам, а посему они теряются, когда последние увольняются или переходят на другую работу.

На большом предприятии лишь подмножество всех имеющихся бизнес-правил относится к любому конкретному приложению. Создание набора общих правил помогает аналитикам требований определить потенциальные требования, которые не были выявлены при обсуждении с пользователями. Наличие главного хранилища бизнес-правил упрощает согласованную реализацию приложений, на которые эти правила влияют.

Правила бизнеса

Согласно определению Business Rules Group (1993), «бизнес-правило — это положение, определяющее или ограничивающее какие-либо стороны бизнеса; его назначение — защитить структуру бизнеса, контролировать или влиять на его операции». Целые методологии разработаны специально для создания и документирования бизнес-правил и их применения в автоматизированных системах (Ross, 1997; von Halle, 2002). Если вы не создаете систему, которая в значительной степени управляется бизнес-правилами, тщательно разработанная методология вам не нужна. Достаточно выявить и задокументировать относящиеся к вашей системе правила и связать их с конкретными функциональными требованиями,

Для организации бизнес-правил предлагается множество разных таксономии (схем классификации) (Ross, 2001; Morgan, 2002; von Halle, 2002). Простейшая из них (рис. 9-1), из пяти типов бизнес-правил, годится в большинстве случаев. Шестая категория — *термины:* важные для бизнеса слова, фразы и аббревиатуры. Их удобно хранить в словаре. Вести согласованный свод бизнес-правил гораздо важнее при разработке продукта, чем горячо дискутировать о том, как их классифицировать. Давайте рассмотрим различные типы бизнес-правил, с которыми вам придется столкнуться.

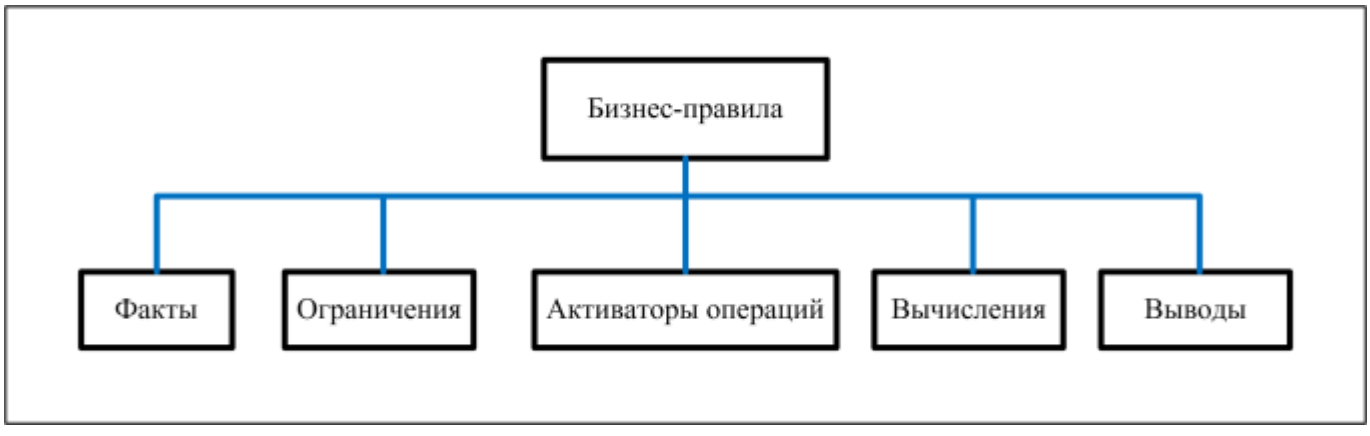


Рис. 9-1. Простая таксономия бизнес-правил

Факты

Факты (facts) — это всего лишь верные утверждения о бизнесе. Зачастую они описывают связи и отношения между важными бизнес-терминами. Факты также называют ***инвариантами*** — неизменными истинами о сущности данных и их атрибутах. Бизнес-правила во многих случаях могут ссылаться на определенные факты, однако последние обычно не преобразуются напрямую в функциональные требования к ПО. Сведения о сущности данных, важных для системы, иногда применяют в моделях данных, создаваемых аналитиком или архитектором БД (подробнее о моделировании данных — в главе 11). Вот примеры фактов:

- на каждый химический контейнер нанесен уникальный штрих-код;
- оплачивается доставка каждого заказа;
- каждый элемент заказа содержит данные о химикате, его качестве, размере контейнера и числе контейнеров;
- стоимость билетов не возвращается, если покупатель изменяет маршрут;
- со стоимости доставки налог с продаж не берется.

Ограничения

Ограничения (constraints) — определяют, какие операции может выполнять система и ее пользователи. Вот некоторые слова и фразы, которые часто применяются при описании ограничивающего бизнес-правила: ***должен, не должен, не может и только***. Например, в таких комбинациях:

- договор о займе человека младше 18 лет должен подписывать один из его родителей или законный опекун;
- постоянный посетитель библиотеки может отложить для себя до 10 книг;
- сотрудник может запросить вещество из списка химикатов первого уровня опасности, только если за последние 12 месяцев он прошел обучающий курс работы с опасными соединениями;
- все программы должны соответствовать правительственным постановлениям, касающимся использования их людьми с ослабленным зрением;
- в корреспонденции не может отображаться больше 4 цифр номера социального страхования гражданина;
- экипажи коммерческих авиарейсов должны каждые 24 часа отдыхать не менее 8 часов.

Так много ограничений

У проектов по разработке ПО множество разных ограничений. Менеджеры проекта должны соблюдать расписание, а также лимиты по персоналу и бюджету. Эти ограничения уровня проекта фиксируются в плане управления проектом по разработке ПО. Ограничения уровня продукта на проектирование и реализацию, уменьшающие доступные разработчику возможности, записаны в спецификации требований к ПО или спецификации архитектуры. Многие бизнес-правила налагают ограничения на деловые операции. Когда бы эти ограничения ни были отражены в функциональных требованиях к ПО, следует указать соответствующие правила в качестве обоснования всех производных требований.

Скорее всего в вашей организации есть политики безопасности, определяющие порядок доступа к информационным системам. Обычно они констатируют, какие пароли следует применять, как часто их надо менять, можно ли применять старые пароли и т.д. Все эти ограничения, касающиеся доступа к приложению, можно считать бизнес-правилами. Ввод каждого такого правила в конкретный код упрощает обновление систем, необходимое для соответствия их новым правилам, например изменение необходимой частоты обновления паролей с 90 до 30 дней.

Как мне отказали

Недавно я хотел заказать, используя часть моих «миль постоянного клиента» авиакомпании Fly-By-Night Airlines, билет для моей жены Крис. Когда я попытался сделать это, сервер FlyByNight.com сообщил мне, что из-за возникшей ошибки выдача билета невозможна, и предложил немедленно позвонить по номеру 800-FLY-NITE. Менеджер по бронированию, который поднял трубку, сообщил, что авиакомпания не может оформить премиальный билет за налетанные мной мили по обычной или электронной почте, поскольку у меня и Крис разные фамилии. Для получения билета мне пришлось ехать в кассу аэропорта и предъявлять удостоверение личности.

Данный инцидент вызван следующим ограничивающим бизнес-правилом: «Если фамилия пассажира отличается от фамилии лица, оформляющего премиальный билет за налетанные мили, лицо, оформляющее билет, должно получить его лично». По-видимому, основание для этого бизнес-правила — предотвращение мошенничества. ПО Web-узла Fly-By-Night исправно реализует данное правило, но создает неудобства клиентам. Вместо того, чтобы сообщить мне о правиле, касающемся разных фамилий и действиях, которые мне следует выполнить, система просто вывела предупреждение об ошибке. В результате мы с менеджером компании Fly-By-Night по бронированию потратили время на ненужный телефонный разговор. Плохо продуманные реализации бизнес-правил могут вызвать неприятие клиентов и, следовательно, отрицательно скажутся на вашем бизнесе.

Активаторы операций

Правило, при определенных условиях приводящее к выполнению каких-либо действий, называется ***активатором операции*** (action enabler). Человек может выполнять эти действия вручную. Как вариант, правило может управлять некоторыми программными функциями, благодаря которым приложение при выполнении определенных условий реализует нужную модель поведения. Условия, определяющие выполнение операции, иногда представляют собой сложную комбинацию значений «истина» и «ложь», выполняющихся для нескольких отдельных условий. Таблица решений, такая, как в главе 11, — это выразительный способ документирования активирующих операции бизнес-правил расширенной логики. Выражение вида «**Если** <*некоторое условие верно или наступило определенное событие*>**, то** <*что-то произойдет*»*,* — это ключ, который описывает активатор операции. Вот несколько примеров таких бизнес-правил:

- если на складе имеются контейнеры с запрошенным химикатом, их следует предложить запросившему химикат лицу;
- если срок хранения контейнера с химикатом истек, об этом необходимо уведомить лицо, у которого в данный момент находится контейнер;
- в последний день квартала должны генерироваться отчеты для Управления по безопасности труда и Агентства по защите окружающей среды о хранении и использовании химикатов в этом квартале;
- если клиент заказал книгу автора, написавшего несколько книг, клиенту следует предложить другие книги этого автора, прежде чем принять заказ.

Выводы

Выводы (inference), иногда эту категорию еще называют предположительным знанием, — это правило, устанавливающее новые реалии на основе достоверности определенных условий. Вывод создает новый факт на основе других фактов или вычислений. Выводы зачастую записывают в формате «если — то», применяемом также при записи бизнес-правил, активирующих операции; тем

не менее, раздел «то» вывода заключает в себе факт или предположение, а не действие. Вот несколько примеров выводов:

- если платеж не поступил в течение 30 календарных дней с момента отправки счета, счет считается просроченным; если поставщик не может поставить заказанный товар в течение пяти дней с момента получения заказа, заказ считается невыполненным;
- считается, что срок хранения контейнеров с химикатами, разлагающимися на взрывоопасные составляющие, истекает через один год с даты изготовления;
- химикаты с токсичностью агента LD50 ниже 5 мг на килограмм массы мыши считаются опасными.

Вычисления

Компьютеры осуществляют вычисления, и поэтому один из классов бизнес-правил определяет ***вычисления*** (computations), выполняемый с использованием математических формул и алгоритмов. Многие вычисления выполняются по внешним для предприятия *правилам,* например по формулам удержания подоходного налога. В отличие от активирующих операции бизнес-правил, для реализации которых иногда приходится создавать специфические функциональные требования к ПО, правила вычислений в той форме, в которой они выражены, можно рассматривать в качестве требований к ПО. Ниже в текстовой форме дано несколько примеров бизнес-правил для вычислений; как вариант, их можно представить в символьной форме, например в виде математического выражения. Представление таких правил в виде таблицы (табл. 9-1) гораздо понятнее, чем длинный список сложных текстовых выражений:

- цена единицы товара снижается на 10% при заказе от 6 до 10 единиц, на 20% — при заказе от 11 до 20 единиц и на 30% — при заказе свыше 20 единиц;
- плата за доставку по суше в пределах страны для заказов весом свыше 2 фунтов составляет $4,75 плюс 12 центов на унцию или ее часть; 1 комиссия за операции с ценными бумагами, осуществлявшиеся в интерактивном режиме, составляет $12 при числе акций от 1 до 5000.
- комиссия за операции, осуществлявшиеся через специалиста по торговле ценными бумагами, составляет $45 при числе акций от 1 до 5000. При числе акций свыше 5000 комиссия составляет половину указанной суммы;
- общая стоимость заказа вычисляется как сумма стоимостей всех заказанных товаров, за вычетом скидок на количество, плюс государственные и местные налоги, действующие в округе, куда будет доставлен товар, плюс стоимость доставки и плюс необязательный страховой сбор.
-

Таблица 9-1. Использование таблицы для вычислений, необходимых для представления бизнес-правил

Идентификатор	Количество приобретаемых единиц товара	Скидка, %
DISC-1	1-5	0
DISC-2	6-10	10
DISC-3	11-20	20
DISC-4	свыше 20	35

Отдельное вычисление может включать множество элементов. В последних примерах общая стоимость учитывает скидку на количество, налог с продаж, стоимость доставки и страховой сбор. Это правило запутанно и сложно для понимания. Чтобы избавиться от данного недостатка, пишите свои бизнес-правила на элементарном уровне, а не объединяйте множество деталей в одно правило. Сложное правило может ссылаться на используемые в нем отдельные правила. Это сделает их краткими и простыми. Кроме того, так вы сможете повторно использовать их и комбинировать различными способами. Чтобы писать вычисления и активирующие операции бизнес-правила на элементарном уровне, не используйте в левой части конструкции «если - то» логику «или», а в

правой части этой же конструкции — логику «и» (von Halle, 2002). К элементарным бизнес-правилам, влияющим на вычисление общей стоимости из примера выше, относятся правила вычисления скидки из табл. 9-1, а также следующие правила:

- если адрес доставки находится в штате, где действует налог на продажи, то он вычисляется исходя из общей стоимости заказанных товаров с учетом скидки;
- если адрес доставки относится к округу, где действует налог на продажи, то он вычисляется исходя из общей стоимости заказанных товаров с учетом скидки;
- страховой сбор составляет 1 % от общей стоимости заказанных товаров с учетом скидки;
- налог на продажи не взимается со стоимости доставки;
- налог на продажи не взимается со страхового сбора.

Эти бизнес-правила называются ***элементарными*** потому, что дальнейшая их детализация невозможна. В конечном итоге вы, скорее всего, создадите множество элементарных бизнес-правил, от комбинаций которых будут зависеть все вычисления и функциональные требования.

Документирование бизнес-правил

Поскольку бизнес-правила зачастую влияют на множество приложений, организациям следует управлять этими правилами на корпоративном уровне, а не на уровне проектов. Для начала достаточно простого каталога бизнес-правил. Большим организациям, а также компаниям, деловые операции и информационные системы которых в значительной степени регулируются и управляются бизнес-правилами, следует создать БД таких правил. Если ваш каталог правил уже не помещается в файле текстового процессора или редактора электронных таблиц или если вы хотите автоматизировать отдельные стороны реализации правил в ваших приложениях, вам наверняка пригодятся коммерческие средства управления правилами. Группа Business Rules Group публикует список продуктов для управления бизнес-правилами на странице *http://www.businessrulesgroup.org/brglink.htm*. По мере того, как вы при работе над приложением определяете новые правила, добавляйте их в каталог, а не вписывайте в документацию конкретного приложения или, что еще хуже, только в его код. При неправильном управлении и выполнении правила, относящиеся к безопасности, защите, финансам и соответствию различным постановлениям, становятся опасными для вас.

Ловушка

Не пытайтесь сделать свой каталог бизнес-правил более сложным, чем необходимо. Используйте простейшую форму документирования правил; это гарантирует, что команда разработчиков сможет эффективно использовать их.

По мере приобретения опыта выявления и документирования бизнес-правил стоит подумать о применении структурированных шаблонов для определения правил разных типов (Ross, 1997; von Halle, 2002). В этих шаблонах описываются образцы ключевых слов и разделов, позволяющих согласованно структурировать правила. Кроме того, они упрощают хранение правил в БД, коммерческих инструментах для управления бизнес-правилами или в ядре правил. Для начала попробуйте использовать простой формат документирования правил, показанный в табл. 9-2 (Kulak и Guiney, 2000).

Как видно из этой таблицы, присвоив каждому правилу идентификатор, вы сможете отслеживать, от какого правила происходит то или иное функциональное требование. Поле «Тип правила» указывает, чем является правило: фактом, ограничением, активатором операции, выводом или вычислением, Поле «Статичное или динамическое» указывает, насколько вероятно изменение правила с течением времени. Источниками правил могут быть корпоративные политики, политики менеджмента, профильные специалисты и прочие лица, документы, например правительственные постановления, а также имеющийся код ПО и определения БД.

Таблица 9-2. Пример каталога бизнес-правил

Идентификатор	Определение правила	Тип правила	Статичное или динамическое	Источник
ORDER-15	Пользователь может запросить вещество из списка химикатов первого уровня опасности, только если за последние **12** месяцев он прошел обучающий курс по работе с опасными соединениями	Ограничение	Статичное	Корпоративная политика
DISC-13	Размер скидки вычисляется на основе размера заказа согласно данным табл. 9-1	Вычисление	Динамическое	Корпоративная политика

Бизнес-правила и требования

Просто задавая пользователем вопрос: «Что представляют собой ваши бизнес-правила?», практически невозможно получить нужную информацию, точно так же, как не удается собрать нужные сведения о требованиях, задавая при сборе требований вопрос пользователям: «Чего вы хотите?» В зависимости от приложения бизнес-правила иногда создаются по ходу работы, а иногда — в процессе обсуждения требований, Зачастую заинтересованные в проекте лица уже знают, какие бизнес-правила влияют на создаваемое приложение, и команда разработчиков будет работать в рамках, определяемых этими правилами, Комплексный процесс выявления бизнес-правил описан Barbara von Halle (2002).

На семинарах для выявления требований аналитик может поинтересоваться обоснованием требований и ограничений, выдвигаемых пользователями. Нередко выясняется, что они определяются бизнес-правилами. Иногда аналитику удается также выявить бизнес-правила, которые определяют прочие артефакты и модели требований (Gottes-diener, 2002). На рис. 9-2 показано несколько возможных источников правил (и в некоторых случаях — вариантов использования и функциональных требований). Кроме того, здесь перечислены некоторые вопросы, которые может задавать аналитик участникам семинара при обсуждении различных проблем и объектов. Следует задокументировать выявленные бизнес-правила, попросить компетентных людей подтвердить их правильность и предоставить недостающие данные.

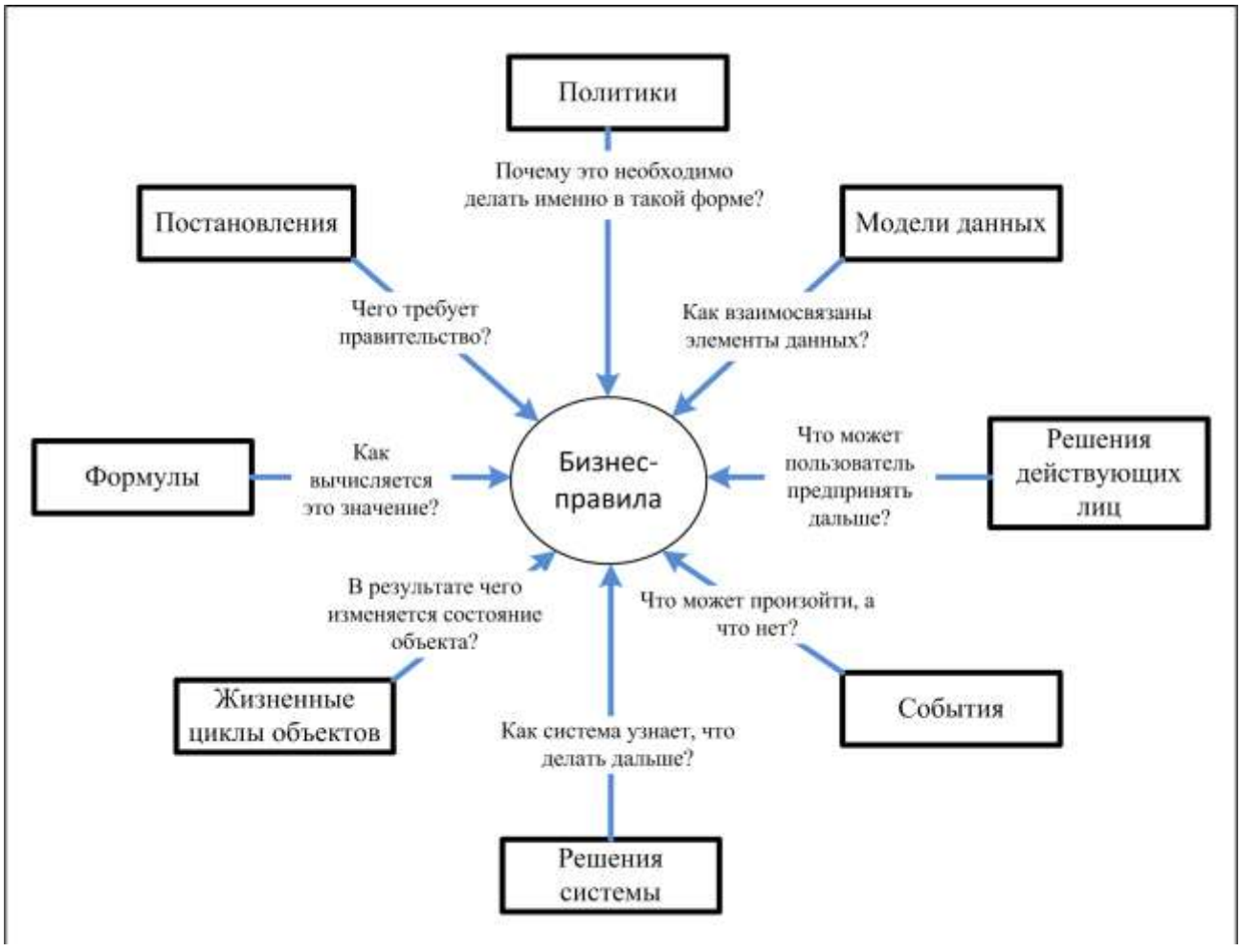


Рис. 9-2. Выявление бизнес-правил с помощью разнообразных вопросов

Идентифицировав и задокументировав бизнес-правила, определите, какие из них необходимо реализовать в программном продукте. Некоторые правила определяют варианты использования и функциональные требования, которые их реализуют. Рассмотрим три правила:

- **Правило № 1 (активатор операции).** «Если срок хранения контейнера с химикатом истек, об этом необходимо уведомить лицо, у которого в данный момент находится контейнер»;
- **Правило № 2 (вывод).** «Считается, что срок хранения контейнеров с химикатами, разлагающимися на взрывоопасные составляющие, истекает через один год с даты изготовления»;
- **Правило № 3 (факт).** «Эфир может спонтанно образовывать взрывоопасную перекись».

Эти правила определяют вариант использования, известный, как «Известить владельца химиката об истечении срока хранения». Одно из функциональных требований для данного варианта использования таково: «Система должна по электронной почте уведомить об владельца контейнера с химикатом об истечении срока хранения химиката».

Связи между функциональным требованием и породившими его бизнес-правилами определяют двумя способами:

- используя атрибут «Источник», указывают правила в качестве источников функционального требования (подробнее — в главе 18);
- определяют связи между функциональным требованием и соответствующими бизнес-правилами в матрице связей (подробнее — в главе 20).

Правила ссылочной целостности данных зачастую реализуют в виде триггеров и хранимых

процедур БД. Такие правила описывают операции по обновлению, вставке и удалению данных, которые должна выполнять система в результате наличия отношений между сущностями данных (von Halle, 2002). Например, если клиент отменил заказ, система должна удалить все пункты, касающиеся неотправленных товаров, входящих в этот заказ.

Иногда бизнес-правила и соответствующие им функциональные требования весьма похожи. Тем не менее, правила — это внешние положения политики, которые требуется реализовать в ПО, следовательно, они управляют функциональностью системы. Каждый аналитик должен решить, какие из имеющихся правил относятся к его приложению, какие необходимо реализовать в нем и как именно.

Вспомним ограничивающее правило для системы контроля химикатов, согласно которому запрос пользователя о выделении опасных химикатов удовлетворяется, только если за последние 12 месяцев пользователь прослушал курс по работе с ними. Аналитику придется определить соответствующие этому правилу функциональные требования в зависимости от того, доступна ли БД записей об обучении в интерактивном режиме или нет. Если да, система может просто просмотреть запись об обучении пользователя и решить, принять или отклонить его запрос. Если записи в интерактивном режиме недоступны, система может временно сохранить запрос на химикат и по электронной почте обратиться к руководителю курса, который, в свою очередь, подтвердит или не подтвердит запрос. Правило одинаково для обеих ситуаций, однако функциональные требования к ПО — действия, предпринимаемые системой, если бизнес-правило встречается в ходе работы — различаются в зависимости от рабочей среды системы.

Во избежание избыточности не дублируйте правила из каталога бизнес-правил в спецификации требований к ПО. Вместо этого спецификация требований к ПО должна ссылаться непосредственно на конкретные правила, как на источник, скажем, алгоритма расчета подоходного налога. Так вы получите несколько преимуществ:

- исчезает необходимость модифицировать и бизнес-правило, и соответствующие функциональные требования при изменении правила;
- в спецификации требований к ПО отражаются последние изменения правил, поскольку спецификация требований к ПО просто ссылается на основную копию правила;
- упрощается повторное использование одного правила в нескольких разделах спецификации требований к ПО и в нескольких проектах безо всякого риска несогласованности, поскольку правила не похоронены в документации отдельного приложения.

Для детального просмотра правил разработчику, читающему спецификацию требований к ПО, придется следовать по перекрестным ссылкам. Это компромисс, на который приходится идти, если вы не собираетесь дублировать информацию. Однако есть одна опасность, связанная с раздельным хранением правил и функциональных требований: правила, имеющие смысл в ограниченном пространстве, при рассмотрении в рабочем контексте совместно с другими правилами могут показаться не такими уж и эффективными. Как и в случае с другими особенностями создания требований, не так просто создать отличное решение для управления бизнес-правилами, которое работало бы во всех ситуациях.

Организации, уверенно управляющие своими бизнес-правилами, очень часто не желают делиться опытом. Они рассматривают наработанные способы использования бизнес-правил для управления ПО как конкурентное преимущество, которого нет у организаций, относящихся к бизнес-правилам небрежно. Когда вы начнете активно выявлять, фиксировать и применять бизнес-правила, всем заинтересованным лицам станет понятнее обоснование тех или иных выбранных вами вариантов разработки приложения.

Что теперь?

- Перечислите все бизнес-правила, которые, по вашему мнению, имеют отношение к вашему текущему проекту. Начните заполнять каталог бизнес-правил, классифицируя правила в соответствии со схемой на рис. 9-1 и обращая особое внимание на источник каждого из них;
- Выясните обоснование каждого из ваших функциональных требований, чтобы выявить дополнительные бизнес-правила;
- Создайте матрицу связей, указывающую, какие функциональные требования и элементы БД реализуют каждое из выявленных вами бизнес-правил.

Глава 10 Документирование требований

Итог разработки требований — задокументированное соглашение между клиентами и разработчиками о создаваемом продукте. Как уже говорилось в предыдущих главах, в документе об образе и границах проекта содержатся бизнес-требования, а пожелания пользователей часто фиксируются в виде вариантов использования продукта. Подробные функциональные и нефункциональные требования к продукту записаны в спецификации к требованиям к ПО. Однако, если вы не соберете все требования вместе в виде структурированного и читабельного материала и не ознакомите с этим документом всех заинтересованных в проекте лиц, то у людей не будет уверенности, что они согласны со всеми положениями.

Способов представления требований несколько:

- документация, в которой используется четко структурированный и аккуратно используемый естественный язык;
- графические модели, иллюстрирующие процессы трансформации состояния системы и их изменения, взаимодействия данных, а также логические потоки, классы объектов и отношения между ними;
- формальные спецификации, где требования определены с помощью математически точных, формальных логических языков.

Последний метод обеспечивает наивысшую степень точности, однако немногие разработчики — и еще меньше клиентов — знакомы с ним, поэтому они не могут, используя этот метод, проверить спецификации на наличие ошибок. Подробно формальные способы составления спецификаций описаны в Davis (1993). Несмотря на недостатки, наиболее практичным подходом остается структурированный естественный язык, дополненный графическими моделями.

Даже самая лучшая спецификация к требованиям не заменит личное обсуждение в ходе проекта. Невозможно заранее выяснить каждый нюанс, используемый при разработке ПО. Старайтесь не прерывать взаимодействие команды разработчиков, представителей клиентов, тестировщиков и других заинтересованных лиц — только так удастся оперативно решать мириады вопросов, которые непременно возникнут.

В этой главе рассматриваются цель, структура и содержание спецификации требований к ПО. Разобраться в материале вам поможет руководство по написанию функциональных требований, а также примеры неверно сформулированных требований и способы их улучшения. В качестве альтернативы традиционным методам документооборота вы можете выбрать базу данных, как коммерческое средство управления требованиями. Это существенно упростит процесс управления, использования и соотнесения требований, однако их качество напрямую зависит от качества информации, которая в них содержится. В главе 21 подробно рассказано об инструментах управления требованиями.

Спецификация требований к ПО

Спецификацию требований к ПО иногда *называют функциональной спецификацией, спецификацией продукта, документ о требованиях и системной спецификацией,* хотя в различных компаниях эти термины понимаются по-разному, В этом документе точно указываются функции и возможности, которыми должно обладать ПО, а также необходимые ограничения. Именно на основе спецификации составляются планы разработки проекта и написания кода, а также особенности тестирования системы и пользовательской документации. Она должна содержать описание поведения системы при различных условиях. Детали дизайна, сборки, тестирования или управления проекта, зафиксированные в спецификации, не должны противоречить ограничениям разработки и развертывания.

Спецификация требований к ПО необходима различным участникам проекта. Клиенты, отдел маркетинга и специалисты по продажам хотят иметь представление о конечном продукте:

- менеджеры проекта по данным спецификации рассчитывают графики, затраты и ресурсы;
- команда разработчиков ПО получает представление о создаваемом продукте;

- группа тестирования составляет планы тестирования, варианты использования и процедуры;
- специалисты по обслуживанию и поддержке получают представление о функциональности каждой составной части продукта;
- составители документации создают руководства для пользователей и окна справки на основании спецификации требований к ПО и дизайна пользовательского интерфейса;
- специалистам, ответственным за обучение персонала, необходима спецификация требований к ПО и документация для пользователей для разработки обучающих материалов;
- персонал, занимающийся юридической стороной проекта, проверяет, соответствуют ли требования к продукту существующим законам и постановлениям;
- субподрядчики строят свою работу и несут юридическую ответственность также согласно спецификации требований к ПО.

Будучи конечным хранилищем требований к продукту, спецификация требований к ПО должна быть ясной и понятной, дабы у разработчиков и клиентов не оставалось ни малейших возможностей для разночтения. Если необходимые функция или качество не включены в соглашение о требованиях, не следует ожидать, что они появятся в конечном продукте.

Не обязательно составлять спецификацию для всего продукта еще до начала разработки, но необходимо зафиксировать требования для каждой составляющей перед ее созданием. Это удобно, если в начале работы участники проекта не смогли определить все требования, а некоторую функциональность необходимо быстро передать в руки пользователей. Однако при работе над любым проектом необходимо достичь согласованности решений по каждому набору требований до начала их реализации разработчиками. Согласованность решений представляет собой процесс изучения и одобрения требований к ПО. При работе с согласованной спецификацией снижается вероятность непонимания и ненужных переделок.

Структурируйте и составляйте спецификацию требований к ПО таким образом, чтобы все заинтересованные в проекте лица смогли в ней разобраться. В главе 1 перечислено несколько характеристик правильно формулированных положений о требованиях и спецификаций. Ниже приведены советы, как сделать требования ясными и понятными:

- разделы, подразделы и отдельные требования должны быть названы согласованно;
- оставьте текст не выровненным по правому краю, не выравнивайте его по ширине;
- оставляйте достаточно свободного пространства («воздуха»);
- используйте средства визуального выделения (такие, как полужирное начертание, подчеркивание, курсив и различные шрифты) последовательно и в разумных пределах;
- создайте оглавление, а также при желании алфавитный указатель, чтобы облегчить пользователям поиск необходимой информации; пронумеруйте все рисунки и таблицы, озаглавьте их и, ссылаясь на них, используйте присвоенные номера;
- если вы ссылаетесь в документе на другие его части, используйте возможности работы с перекрестными ссылками в вашем редакторе, а не сложную кодировку страниц;
- применяйте гиперссылки, чтобы читатель смог быстро перейти к соответствующим разделам спецификации или другим документам; для структурирования необходимой информации используйте соответствующий шаблон.

Требования к именованию

Чтобы легко отслеживать и модифицировать материал, каждое функциональное требование должно быть представлено уникально и неизменно. Это позволит вам ссылаться на определенные требования в запросе на изменения, в хронологии изменений, в перекрестных ссылках или матрице для отслеживания требований. При этом также упрощается многократное использование требований в нескольких проектах. Маркированные списки для этих целей не годятся. Давайте взглянем на достоинства и недостатки некоторых методов именования требований. Выберите любой метод, наиболее подходящий вам.

Нумерация по порядку

Это способ, при котором каждому требованию присваивается уникальный порядковый номер,

например UR-9 или SRS-43. Коммерческие средства управления требованиями присваивают такой идентификатор, когда пользователь добавляет новое требование в БД. (Эти средства также поддерживают функцию иерархического именования.) Префикс обозначает тип требования, например UR означает **«user requirement»** («пользовательское требование»). При удалении требования номер повторно не используется. Такой простой поход нумерации не обеспечивает логического или иерархического группирования связанных требований, а названия не раскрывают содержания требования.

Иерархическая нумерация

Это наиболее распространенный способ. Если функциональные требования приводятся в разделе 3.2 спецификации, то все их номера будут начинаться с 3.2 (например, 3.2.4.3). Чем больше цифр, тем больше уровень детализации требования. Это способ отличается простотой и компактностью. Ваш текстовый редактор может назначить номера автоматически. Однако даже в спецификации среднего размера нумерация может быть весьма длинной. Вдобавок названия, состоящие только из цифр, ничего не говорят о назначении требований. Когда же вы начнете применять этот метод на практике, то обнаружите, что присваиваемые названия не являются неизменными. Если вы вставите новое положение, то номера всех последующих фрагментов увеличатся. А после удаления или перемещения требования — уменьшатся. При этом нарушаются все ссылки на эти фрагменты.

Ловушка

Однажды один аналитик заметил: «Мы не разрешаем людям вставлять требования — сбивается нумерация». Не позволяйте применять неэффективные приемы, которые могут препятствовать эффективной и разумной работе.

Вы можете усовершенствовать этот способ. Пронумеруйте важнейшие разделы требований иерархически, а затем выделите отдельные функциональные требования в каждом разделе с помощью короткого текстового кода, после которого проставьте порядковый номер. Например, если в спецификации требований к ПО есть «Раздел 3.5 — Функции редактора», то требования в нем будут названы ФР-1, ФР-2 и т.д. При этом соблюдается определенная иерархия и структурированность документа, названия достаточно короткие, осмысленные и менее зависят от местоположения в документе.

Иерархические текстовые тэги

Консультант Tom Gilb предложил текстовую схему иерархических тэгов именования отдельных требований (Gilb, 1988). Рассмотрим такое требование: «Система должна запрашивать у пользователя подтверждение запроса, когда тот хочет печатать более 10 копий». Этому требованию можно присвоить тэг Print.ConfirmCopies, который означает, что это требование является частью функции печати и связано с количеством печатаемых копий. Иерархические текстовые тэги структурированы, их названия осмысленны и не зависят от добавления, удаления или перемещения остальных положений. Недостатком является то, что они более громоздки, чем цифровые элементы, однако это незначительная плата за стабильность.

Когда информации недостаточно

Иногда вы вдруг понимаете, что вам не хватает определенной информации о конкретном требовании. Ничего не поделаешь, придется проконсультироваться с клиентом, проверить описание внешнего интерфейса или создать прототип. Для того чтобы пометить пробелы в данных, используйте пометку «TBD» (***to be determined*** — необходимо определить).

Выясните все до реализации набора требований. Любые неясности повышают риск ошибки со стороны разработчика или тестера и, как следствие, вероятность переделки. Если разработчику не хватает информации, он не всегда обращается для разрешения проблемы к аналитику требований. Иногда он пытаться снять вопрос самостоятельно, реализуя лучший с его точки зрения вариант, который не всегда корректен для проекта в целом. Если же необходимо приступить к сборке, а

неразрешенные вопросы еще остаются, следует или отложить реализацию неясных требований, или сделать эти части продукта легко модифицируемыми.

Ловушка

Неясности не решаются сами собой. Запишите, кто отвечает за каждый вопрос, как и когда. Пронумеруйте все пометки «TBD», это упростит контроль.

Пользовательские интерфейсы и спецификация требований к ПО

Включение элементов пользовательского интерфейса в спецификацию имеет как преимущества, так и недостатки. Отрицательным моментом можно считать то, что изображения и архитектура пользовательского интерфейса отображают решения (дизайн), а не требования. Откладывая согласование решений в спецификации требований к ПО до завершения разработки пользовательского интерфейса, вы можете заставить нервничать людей, которые и так обеспокоены тем, что на работу над требованиями затрачено слишком много времени. Включение элементов пользовательского интерфейса в требования может сместить приоритеты, и описание функциональности окажется неполной.

Макет экрана не заменит пользовательских и функциональных требований. Не следует ожидать, что разработчики смогут сделать вывод о базовой функциональности и взаимосвязи данных по моментальным снимкам экрана. У одной компании, создающей ПО для Интернета, постоянно возникали одни и те же проблемы из-за того, что разработчики непосредственно переходили к работе над визуальным дизайном сразу же после подписания контракта. У них не было ясного представления о том, как пользователи будут работать с Web-сайтом, поэтому им приходилось тратить массу времени на последующие исправления.

Из положительных сторон следует отметить, что изучение возможных пользовательских интерфейсов (таких, как рабочий прототип) делает требования более осязаемыми и для пользователей, и для разработчиков. Изображения пользовательского интерфейса помогают при планировании и оценке проекта. Подсчет элементов ***графического интерфейса пользователя*** (graphical user interface, GUI) или числа ***функциональных точек*** [1] (function points), связанных с каждым экраном, позволяет оценить размер проекта и, следовательно, затраты на реализацию.

«Золотая середина» подразумевает включение концептуальных изображений — набросков — в спецификацию требований к ПО без обязательного точного соблюдения этих моделей при реализации. При этом улучшается взаимодействие специалистов без ненужных ограничений для разработчиков. Например, предварительный набросок сложного диалогового окна может проиллюстрировать назначение части требований, однако опытный визуальный дизайнер сумеет превратить его в диалоговое окно с вкладками для удобства работы пользователя.

Шаблон спецификации требований к ПО

Каждая организация, специализирующаяся на разработке ПО, должна принять один или несколько стандартных шаблонов спецификации требований к ПО для использования в проектах. Доступны различные шаблоны спецификации (Davis, 1993; Robertson и Robertson, 1999; Leffingwell и Widring, 2000). Многие применяют шаблоны, созданные на основе того, что описан в IEEE Standard 830-1998, «IEEE Recommended Practice for Software Requirements Specifications» (IEEE, 1998b). Он годится для самых разных проектов, однако в нем встречаются ограничения и неясные места. Если вы беретесь за проекты различных типов и размеров, от конструирования новой объемной системы до небольших улучшений уже работающих систем, заведите для проектов каждого крупного класса отдельный шаблон спецификации.

---

[1] Функциональной точкой называется единицаизмерений доступных пользователем функций приложения независимо от того, как они сконструированы. Вы можете оценить функциональны ли точки, исходя из требований пользователя, по числу внешних логических файлов и элементов вводимых данных, выводимых данных и запросов (IFPUG, 2002).

1. Введение
- 1.1 Назначение
- 1.2 Соглашения, принятые документах
- 1.3 Предполагаемая аудитория и рекомендации по чтению
- 1.4 Границы проекта
- 1.5 Ссылки

2. Общее описание
- 2.1 Общий взгляд на продукт
- 2.2 Особенности продукта
- 2.3 Классы и характеристики пользователей
- 2.4 Операционная среда
- 2.5 Ограничения проектирования и реализации
- 2.6 Документация для пользователей
- 2.7 Предположения и зависимости

3. Функции системы
- 3.x Функция системы X
  - 3.x.1 Описание и приоритеты
  - 3.x.2 Последовательности «воздействие - реакция»
  - 3.x.3 Функциональные требования

4. Требования к внешнему интерфейсу
- 4.1 Интерфейсы пользователя
- 4.2 Интерфейсы оборудования
- 4.3 Интерфейсы ПО
- 4.4 Интерфейсы передачи информации

5. Другие нефункциональные требования
- 5.1 Требования к производительности
- 5.2 Требования к охране труда
- 5.3 Требования к безопасности
- 5.4 Атрибуты качества

6. Остальные требования
Приложение А. Словарь терминов
Приложение Б. Модели анализа
Приложение Г. Список вопросов

Рис. 10-1. Шаблон для спецификации требований к ПО

На рис. 10-1 показан шаблон спецификации требований к ПО, созданный на основе стандарта IEEE 830; в нем предлагается множество примеров дополнительных требований к продукту, которые вы можете включить в свою спецификацию. В приложении Г показан пример спецификации требований к ПО, производной от этого шаблона. Измените его в соответствии с особенностями вашего проекта. Если какой-то раздел вашего шаблона не годится для конкретного проекта, не удаляйте его заголовок, но укажите, что он неприменим. В этом случае у пользователя не возникнет подозрения, что что-то важное было пропущено по невнимательности. Если вам постоянно приходится пропускать одни и те же разделы, это означает, что шаблон следует настроить. Создайте оглавление и журнал изменений спецификации требований к ПО, где указаны дата изменения, сотрудник, внесенное изменение и ее причина. Иногда фрагмент информации логически подходит для нескольких разделов шаблона. Гораздо важнее аккуратно и последовательно фиксировать информацию, чем горячо обсуждать, где следует хранить каждый элемент.

Сравните рис. 5-2 и 10-1. Вы увидите, что шаблон спецификации требований к ПО и шаблон документа об образе и границах в некоторых местах перекрываются (например, общие элементы есть в границах проекта, в описании особенностей продукта и в разделах операционной среды). Причина кроется в том, что вы создали только один документ, касающийся требований. Если вы используете

оба шаблона измените их таким образом, чтобы ликвидировать избыточность, при необходимости объединив разделы. Возможно, соответствующие разделы спецификации требований к ПО пригодятся для детализации определенного прогноза или высокоуровневой информации, содержащихся в документе об образе и границах проекта. Если же вы решите вырезать и вставить фрагменты из одного документа в другой, то возможна опасность того, что в обоих появятся ненужные повторы.

В следующих разделах этой главы рассказано, какую информацию следует включать в каждый раздел спецификации требований к ПО. Вы можете объединить материал с помощью ссылки на другие документы (например, об образе и границах проекта или спецификацию интерфейса), а не дублировать его в спецификации требований к ПО.

1.Введение

Введение представляет собой обзор, помогающий читателям разобраться в структуре и принципе использования спецификации требований к ПО.

1.1 Назначение

Определите продукт или приложение, требования для которого указаны в этом документе, в том числе редакцию или номер выпуска. Если эта спецификация требований к ПО относится только к части системы, идентифицируйте эту часть или подсистему.

1.2 Соглашения, принятые в документах

Опишите все стандарты или типографические стандарты, включая стили текста, особенности выделения или замечания. Например, укажите, унаследован ли приоритет, указанный для требований высшего уровня, всеми их детализированными требованиями, или каждое положение о функциональных требованиях должно обладать собственным приоритетом.

1.3 Предполагаемая аудитория и рекомендации по чтению

Перечислите пользователей, для которых предназначена эта спецификация требований к ПО. Опишите содержание документа и его структуру. Порекомендуйте наиболее подходящую для каждого класса читателей последовательность чтения документа.

1.4 Границы проекта

Кратко опишите ПО и его назначение. Покажите, как связан продукт с пользователями или корпоративными целями, а также с бизнес-целями и стратегиями. Если имеется отдельный документ об образе и границах проекта, не повторяйте его содержимое, а просто сошлитесь на него. Если спецификацию требований к ПО предполагается разрабатывать постепенно, она должна содержать собственное положение об образе и границах продукта в качестве подраздела долгосрочного стратегического образа.

1.5 Ссылки

Перечислите все документы или другие ресурсы, на которые вы ссылаетесь в этой спецификации, в том числе гиперссылки на них. Это могут быть руководства по стилям пользовательского интерфейса, контракты, стандарты, спецификации к системным требованиям, документы о вариантах использования, спецификации интерфейса, концептуальные документы и спецификация требований к ПО для продуктов, на которые вы ссылаетесь. Объем информации должен быть достаточным для того, чтобы пользователь сумел при необходимости получить доступ к каждому указанному материалу, а именно: название, имя автора, номер версии, дата и источник или расположение (например, сетевая папка или URL).

2. Общее описание

В этом разделе представлен общий обзор продукта и среды, в которой он будет применяться, предполагаемая пользовательская аудитория, а также известные ограничения, предположения и зависимости.

2.1 Общий взгляд на продукт

Опишите содержание и происхождение продукта. Поясните, является он новым членом растущего семейства продуктов, новой версией существующей системы, заменой существующего приложения или совершенно новым продуктом? Если спецификация требований определяет компонент более крупной системы, укажите, как это ПО соотносится со всей системой и определите основные интерфейсы между ними.

2.2 Особенности продукта

Перечислите основные особенности продукта или его главные функции. Детали будут изложены в разделе 3 спецификации требований к ПО, здесь же следует их только указать. Также здесь уместно проиллюстрировать основные группы требований и их взаимоотношения, например показать диаграмму потоков данных высшего уровня, диаграмму вариантов использования или диаграмму классов.

2.3 Классы и характеристики пользователей

Определите различные классы пользователей, которые, как предполагается, будут работать с вашим продуктом, и опишите их соответствующие характеристики (см. главу 6). Некоторые требования могут относиться только к определенным классам пользователей. Определите привилегированные классы пользователей. Классы пользователей представляют подмножество заинтересованных в проекте лиц, их описание приводится в документе об образе и границах проекта.

2.4 Операционная среда

Опишите рабочую среду ПО, включая аппаратные средства, операционные системы и их версии, а также географическое местоположение пользователей, серверов и баз данных. Перечислите все остальные компоненты ПО или приложений, с которыми система должна быть совместима. В документе об образе и границах проекта эта информация может быть раскрыта более подробно.

2.5 Ограничения проектирования и реализации

Опишите любые факторы, которые ограничат возможности, доступные разработчикам, и логически обоснуйте каждое положение. Ограничения могут быть такого рода:

- определенные технологии, средства, языки программирования и базы данных, которые следует использовать или избегать;
- ограничения, налагаемые операционной средой продукта, например типы и версии установленных Web-браузеров;
- обязательные соглашения или стандарты разработки (например, если обслуживать ПО будут клиенты, то они должны указать особенности дизайна и стандарты программирования, которые субподрядчик обязан соблюдать);
- обратная совместимость с продуктами, выпущенными ранее;
- ограничения, налагаемые бизнес-правилами (они должны быть зафиксированы в других документах, как рассказано в главе 9);
- ограничения, связанные с оборудованием, например требования к срокам, ограничения памяти или процессора, размер, вес, материалы или затраты;
- соглашения, связанные с пользовательским интерфейсом существующего продукта, которые необходимо соблюдать при улучшении существующего продукта;
- стандартный формат обмена данными, например XML.

2.6 Документация для пользователей

Перечислите все компоненты пользовательской документации, поставляемые с исполняемым ПО. В них могут входить руководства пользователя, онлайн-справка и обучающие программы. Определите все необходимые форматы, стандарты и средства поставки документации.

2.7 Предположения и зависимости

Предположением (assumption) называется положение, которое считается истинным при отсутствии доказательства или определяющей информации. Проблемы возможны в том случае, если предположение неверны, не находятся в совместном использовании или они изменяются, поэтому определенные предположения можно отнести к группе рисков проекта. Один пользователь спецификации может считать, что продукт будут соответствовать особому стандарту пользовательского интерфейса, тогда как другой предположит нечто совершенно иное. Разработчик может думать, что определенный набор функций написан специально для этого приложения, аналитик— что он будет взят из предыдущего проекта, а менеджер проекта— что предполагается приобрести коммерческую библиотеку функций.

Определите все ***зависимости*** (dependencies) проекта от внешних факторов, такие, как дату выпуска следующей версии операционной системы или выпуск промышленного стандарта. Если вы планируете встроить в систему компоненты, разрабатываемые в другом проекте, то вы зависите от своевременной их поставки. Если эти зависимости уже где-то задокументированы, например, в плане

проекта, сошлитесь здесь на них.

3. Функции системы

Шаблон на рис. 10-1 структурирован с помощью особенностей системы — это еще один способ систематизации функциональных требований. Другие методы классификации: по вариантам использования, режиму работы, классам пользователей, стимулам, реакциям, классам объектов или функциональной иерархии (IEEE, 1998b). Возможны также комбинации этих элементов, например, варианты использования внутри классов пользователей. Не существует единственно правильного метода организации; выберите тот, при котором пользователям будет легче понять предполагаемые возможности продукта. Я опишу схему особенностей на примере.

3.x Функция системы X

Укажите название особенности несколькими словами, например «3.1 Проверка правописания». Так же назовите подразделы с 3.x.1 по 3.x.3 для каждой функции системы.

3.x.1 Описание и приоритеты

Кратко опишите особенность функции и укажите, обладает ли она высоким, средним или низким приоритетом (см. главу 14.) Приоритеты являются динамической характеристикой, они могут изменяться в ходе проекта. Если вы используете средство управления требованиями, определите атрибут требований для приоритета. Атрибуты требований обсуждаются в главе 18, а средства управления требованиями — а главе 21.

3.x.2 Последовательности «воздействие - реакция»

Перечислите последовательность воздействий, оказываемых на систему (действия пользователей, сигналы внешних устройств и др.), и отклики системы, определяющие реакцию конкретной функции. Эти воздействия соответствуют первоначальным шагам для вариантов использования или внешним системным событиям.

3.x.3 Функциональные требования

Перечислите по пунктам детализированные функциональные требования, которые связаны с этой особенностью. Здесь должны быть представлены определенные характеристики ПО, чтобы пользователь мог задействовать эту функцию или реализовать варианты использования. Опишите, как продукт должен реагировать на ожидаемые ошибки, неправильный ввод информации или неверные действия. Присвойте каждому функциональному требованию уникальное имя.

4. Требования к внешнему интерфейсу

По мнению Richard Thayer (2002), «требования к внешнему интерфейсу определяют оборудование, ПО или элементы баз данных, с которыми система или компонент должны взаимодействовать...» Информация этого раздела позволяет вам быть уверенным, что система будет должным образом взаимодействовать с внешними компонентами. Если у разных частей продукта разные внешние интерфейсы, вставьте подобный раздел в детализированные требования для каждой такой части.

Выработка согласованного решения, касающегося внешнего и внутреннего интерфейса системы, признана наилучшим приемом в области разработки ПО (Brown, 1996). Вставьте подробные описания компонентов данных и элементов управления интерфейса в словарь данных. В сложной системе с множеством подкомпонентов следует использовать раздельные спецификации для интерфейсов или спецификацию системной архитектуры (Hooks и Farry, 2001). В документацию по интерфейсу можно включить ссылки на материал из других документов. Например, ссылка на спецификацию программного интерфейса другого приложения (API) или на руководство по работе с устройством, где перечислены коды с ошибками, которые устройство может отправить ПО.

Войны интерфейсов

Две команды разработчиков ПО объединились для создания флагманского продукта Datum Corporation. Команда, отвечающая за базу знаний, создала ядро сложного интерфейса на C++, а команда, отвечающая за приложения, реализовала пользовательский интерфейс на Microsoft Visual Basic. Обе подсистемы взаимодействовали между собой посредством API. К сожалению, команда, отвечающая за базу знаний, периодически модифицировала API, в результате систему не удавалось собрать и запустить на выполнение должным образом. Команде, отвечающей за приложения, потребовалось несколько часов, чтобы распознать каждую проблему и определить основную причину — изменение API. Эти изменения не согласовывались, не доводились до сведения всех заинтересованных в проекте лиц и не были координированы с соответствующими модификациями в коде, написанном на Visual Basic. Интерфейс скрепляет компоненты вашей системы, включая пользователей, поэтому необходимо документировать детали интерфейса и синхронизировать модификации в процессе контроля за изменениями вашего проекта.

4.1 Интерфейсы пользователя

Опишите логические характеристики каждого пользовательского интерфейса, который необходим системе. Некоторые из них перечислены здесь:

- ссылки на стандарты графического интерфейса пользователей или стилевые рекомендации для семейства продукта, которые необходимо соблюдать;
- стандарты шрифтов, значков, названий кнопок, изображений, цветовых схем, последовательностей полей вкладок, часто используемых элементов управления и т.п.; конфигурация экрана или ограничения разрешения;
- стандартные кнопки, функции или ссылки перемещения, одинаковые для всех экранов, например кнопка справки;
- быстрые клавиши;
- стандарты отображения сообщений;
- стандарты конфигурации для упрощения локализации ПО;
- специальные возможности для пользователей с проблемами со зрением.

Детально документируйте детали пользовательского интерфейса, такие, как конфигурации определенных диалоговых окон, в отдельной спецификации пользовательского интерфейса, а не в спецификации требований к ПО. Конечно, добавить еще одну точку зрения на требования полезно, но необходимо подчеркнуть, что эти модели не являются официальными планами экрана. Если в спецификации требований к ПО говорится об улучшении существующей системы, то стоит включить в документ изображения экранов в том виде, как они будут реализованы. Для разработчиков уже заданы ограничения существующей системой, поэтому можно заранее представить, как будут выглядеть измененные, а, возможно, и новые экраны.

4.2 Интерфейсы оборудования

Опишите характеристики каждого интерфейса между компонентами ПО и оборудования системы. В описание могут входить типы поддерживаемых устройств, взаимодействия данных и элементов управлений между ПО и оборудованием, а также протоколы взаимодействия, которые будут использоваться.

4.3 Интерфейсы ПО

Опишите соединения продукта и других компонентов ПО (идентифицированные по имени и версии), в том числе базы данных, операционные системы, средства, библиотеки и интегрированные коммерческие компоненты. Укажите назначение элементов сообщений, данных и элементов управления, обмен которыми происходит между компонентами ПО. Опишите службы, необходимые

внешним компонентам ПО, и природу взаимодействия между компонентами. Определите данные, к которым будут иметь доступ компоненты ПО. Если механизм предоставления общего доступа к данным должен быть реализован определенным способом, например в качестве глобальной области данных, то укажите его как ограничение.

4.4 Интерфейсы передачи информации

Укажите требования для любых функций взаимодействия, которые будут использоваться продуктом, включая электронную почту, Web-браузер, протоколы сетевого соединения и электронные формы. Определите соответствующие форматы сообщений. Опишите особенности безопасности взаимодействия или шифрования, частоты передачи данных и механизмов синхронизации.

5. Другие нефункциональные требования

В этом разделе описываются остальные нефункциональные требования, не относящиеся к требованиям к интерфейсу, которые представлены в разделе 4, и к ограничениям, описываемым в разделе 2.5.

5.1 Требования к производительности

Укажите специальные требования к производительности для различных системных операций. Обоснуйте их необходимость для того, чтобы помочь разработчикам принять правильные решения, касающиеся проектирования. Например, из-за жестких требований к времени отклика базы данных разработчики могут зеркализовать базу данных в нескольких географических метаположениях или денормализовать связанные таблиц баз данных для получения более быстрого ответа на запрос вашего проекта. Пропустите этот раздел, если все необходимые требования уже расписаны в других разделах.

Приложение А. Словарь терминов

Поясните термины, которые пользователю необходимо знать для правильного понимания спецификации требований к ПО, включая сокращения и аббревиатуры. Расшифруйте каждое сокращение и приведите его определение. Подумайте о создании расширенного словаря для нескольких проектов. В этом случае в спецификации требований к ПО будут определены только те термины, которые относятся лишь к данному проекту.

Приложение Б. Модели анализа

В этом необязательном разделе описывается, а точнее напоминается о таких моделях анализа, как диаграммы потока данных, диаграммы классов, диаграммы перехода состояния и диаграммы «сущность - связь» (см. главу 11)

Приложение В. Список вопросов

Это динамический список еще не разрешенных проблем, связанных с требованиями. Это могут быть элементы, помеченные как «TBD» (***to be determined*** — необходимо определить), отложенные решения, необходимая информация, неразрешенные конфликты и т.п. Все это не обязательно включать в спецификацию требований к ПО, но в некоторых организациях принято прилагать список «TBD» к спецификации требований к ПО. Постарайтесь как можно быстрее разрешить эти проблемы, чтобы они не стали препятствием к своевременному созданию основ спецификации требований к ПО высокого качества.

Принципы создания требований

Не существует выверенного способа написания идеальных требований, а лучший учитель — это опыт, который нарабатывается со временем. Безупречную документацию по требованиям отличает технический стиль изложения и пользовательская терминология, а не компьютерный сленг. Kovitz (1999) предлагает множество рекомендаций и примеров на эту тему. Вот некоторые из них:

- используйте полные предложения, с правильной грамматикой, правописанием и пунктуацией. Предложения и абзацы должны быть краткими и ясными;
- используйте действительный залог (например, «Система сделает то-то», а не «Произойдет то-то);
- последовательно используйте термины и именно так, как они определены в словаре. Остерегайтесь синонимов и слов, близких по значению. Не следует в спецификации требований к ПО пытаться разнообразить лексику, чтобы заинтересовать читателя;
- нечеткие требования верхнего уровня следует детализировать таким образом, чтоб они стали абсолютно ясны;
- требования следует излагать последовательно, например «Система будет» или «Пользователь будет», затем — активный глагол, а после — наблюдаемый результат. Укажите инициирующие условия или действия, вследствие которых система ведет себя определенным образом. Например, «Если запрошенный химикат найден на складе химикатов, система отобразит список всех хранимых на складе контейнеров с указанным химикатом». Вы можете использовать «должно» как синоним «будет», однако следует избегать «следовало бы», «может», «можно было бы» и аналогичных слов, из которых не ясно, необходимо ли действие;
- при указании требования в форме «Пользователь будет...» идентифицируйте определенного исполнителя (например, «Покупатель будет...»);
- применяйте списки, рисунки, графики и таблицы, чтобы представить информацию визуально. Читателей утомляет большой объем сплошного текста;
- подчеркните наиболее значимые фрагменты информации. Здесь годятся: графики, последовательности, в которых первый элемент подчеркивается, повторы, пустое пространство и визуальный контраст, например затенения (Kovitz, 1999);
- требования, изложенные неясным языком, не поддаются проверке, поэтому избегайте двусмысленных и субъективных терминов, В табл. 10-1 перечислены некоторые из них, а также приводятся рекомендации, как исправить такие неясности.

Ловушка

Оценка качества требований зависит от того, кто их оценивает. Аналитик может верить, что создал документ абсолютно ясный, безо всяких неясностей и проблем. Однако если у читателей возникли вопросы, значит, требуется доработка.

Таблица 10-1. Неоднозначные термины, которых следует избегать в спецификаций к требованиям

Неоднозначные термины	**Способы их улучшения**
Приемлемый, адекватный	Определите, что понимается под приемлемостью и как система это может оценить
Практически выполнимо	Не заставляйте разработчиков определять, что под этим понимается. Поставьте пометку «TBD» и определите дату, к которой эту проблему следует разрешить
По меньшей мере, как минимум, не более чем, не должно превышать	Укажите минимальное и максимальное допустимые значения

Между	Определите, указаны ли конечные точки
Зависит от	Определите природу зависимости. Обеспечивает ли другая система ввод данных в вашу систему, надо ли установить другое ПО до запуска вашей системы и зависит ли ваша система от другой при выполнении определенных расчетов или служб?
Эффективный	Определите, насколько эффективно система использует ресурсы, насколько быстро она выполняет определенные операции и насколько она проста в обращении
Быстрый, моментальный	Укажите минимальную приемлемую скорость, при которой система выполняет определенное действие
Гибкий	Опишите способы изменения системы в ответ на изменения условий или бизнес-потребностей
Улучшенный, лучший, более быстрый, превосходный	Определите количественно, насколько лучше или быстрее стали показатели в определенной функциональной области
Включает; включает в себя, но не огранечен этим; и т.д.; и т.п.; такой как	Список элементов должен включать все возможности. В противном случае его нельзя использовать при разработке или тестировании
Максимизируйте, минимизируйте, оптимизируйте	Укажите минимальное и максимальное допустимые значения определенного параметра
Обычно, в идеальном варианте	Также опишите поведение системы при нештатных или неидеальных условиях
Необязательно	Укажите, кто делает выбор: системы, пользователь или разработчик
Разумный, при необходимости, при соотвествующих условиях	Объясните, как это выполнить
Устойчивый к сбоям	Определите, как система должны обрабатывать исключения и реагировать на неожиданные условия работы
Цельный, прозрачный, элегантный	Выразите ожидания пользователя, применяя характеристики продукта, которые можно наблюдать
Несколько	Укажите сколько или задайте минимальную и максимальную границы диапазона
Не следует	Старайтесь формулировать требования в позитивной форме, описывая, что именно система будет делать
Реальный	Поясните этот термин
Достаточный	Укажите, какая степень чего-либо свидетельствует **о** достаточности
Поддерживает, позволяет	Дайте точное определение, какие действия система будет выполнять, поддерживая конкретную возможность

Дружественный, простой, легкий	Опишите системные характеристики, которые будут отвечать потребностям пользователей и его ожиданиям, касающимся легкости и простоты применения продукта

Составляйте требования настолько подробно, чтобы при удовлетворении требования задача клиента была бы выполнена, однако избегайте ненужных ограничений разработки. Требования должны быть сформулированы достаточно подробно, чтобы риск непонимания был минимальным, для этого необходимо учесть знания и опыт разработчиков. Если разработчики предлагают несколько способов удовлетворения требования и все они приемлемы, значит, особенности продукта и детализация изложения выбраны верно. Точно сформулированные требования повышают вероятность того, что ожидания клиентов будут реализованы; менее строгие формулировки дают разработчикам больше свободы для интерпретаций. Если разработчику по составленной спецификации не удается абсолютно ясно представить себе ожидания клиентов, следует включить дополнительную информацию, чтобы снизить риск последующих исправлений.

Создатели документации зачастую тратят массу сил, чтоб «поймать» нужный уровень детализации. Попробуйте отдельно описать требования, которые можно протестировать. Если вам удастся придумать несколько вариантов тестирования, значит, необходимый уровень детализации достигнут. Если ваши тесты многочисленны и разнообразны, вероятно, несколько требований соединены вместе; их следует разделить на более простые. Тестируемые требования были предложены в качестве атрибута размера ПО (Wilson, 1995).

При написании требований соблюдайте уровень детализации. Мне приходилось видеть в одной и той же спецификации положения, которые значительно варьировались в их границах. Например, «Комбинация клавишей Ctrl+S будет интерпретироваться как «Сохранить файл» и «Комбинация клавишей Ctrl+P будет интерпретироваться как «Печать файла» считались отдельными требованиями, а «Продукт будет реагировать на команды редактирования, введенные голосом» описывалось как целая подсистема (или даже продукт!), а не одно функциональное требование.

Избегайте длинных повествовательных абзацев, которые содержат несколько требований. Наличие в требовании таких слов, как «и», «или» и «также», предполагает, что несколько требований могли быть объединены. Это не означает, что вы не можете использовать союз «и», но если вы делаете это, проверяйте, соединяет ли он две части одного требования или два отдельных требования. Никогда не используйте «и/или» в требованиях; это оставляет читателю свободу маневра. Такие выражения, как «пока не» и «кроме» также указывают на наличие нескольких требований: «Кредитная карточка покупателя будет считаться действительной для платежей до тех пор, пока не истечет ее срок действия». Разделите это положение на два — для двух условий, когда срок действия кредитной карты истек, и когда она действительна.

Избегайте излишних положений о требованиях. Повторяя требований в нескольких местах спецификации требований к ПО, вы облегчаете чтение документа, но затрудняете его поддержку. Одинаковые требования придется изменять одновременно, в противном случае возникнет несогласованность. Перекрестные ссылки на связанные между собой элементы в спецификации помогут вам синхронизировать последние при внесении изменений. Сохраняя отдельные требования в средстве управления требованиями или в базе данных только один раз, вы решите проблему избыточности и упростите повторное использование требований, общих для нескольких проектов.

Попытайтесь найти наиболее эффективный метод представления каждого требования. Как-то я проверял спецификацию требований к ПО, в которой содержался набор требований, предлагаемый в качестве образца: «Текстовый редактор сможет проанализировать документы следующего ***<формат>***, определяющих постановления следующей ***<юрисдикции>***». Предлагалось три возможных значения ***<формат>*** и четыре возможных значения ***<уровень юрисдиции>*** для двенадцати схожих требований. Однако из этих двенадцати одного не хватало, а одно повторялось. Найти ошибку можно было одним единственным образом — составить таблицу всех возможных вариантов и проверить их. Ошибку удалось бы предотвратить, если требования в спецификации были составлены по образу и подобию табл. 10-2. Требования более высокого уровня могут звучать так: «ТР-13 Текстовый редактор сможет провести анализ документов нескольких форматов, определяющих постановления на уровнях юрисдикции, как показано в табл. 10-2 ». Если у какой-либо комбинации нет соответствующего функционального требования, укажите в ячейке таблицы Н/П (не применимо).

Таблица 10-2. Образец табличного формата для перечисления номеров требований, предлагаемых в качестве образца

Уровень юрисдикции	Теговый формат	Безтеговый формат	Формат ASCII
Федеральный	ТР-13.1	ТР-13.2	ТР-13.3
Региональный [на уровне штата]	ТР-13.4	ТР-13.5	ТР-13.6
Территориальный (местный)	ТР-13.7	ТР-13.8	ТР-13.9
Международный	ТР-13.10	ТР-13.11	ТР-13.12

Примеры требований: до и после

В главе 1 перечислено несколько характеристик качественно сформулированных требований: полные, правильные, выполнимые, необходимые, имеющие приоритет, точно выраженные и поддающиеся проверке. Поскольку наличие требований, не отвечающих этим характеристикам, приводит к неразберихе, напрасно затраченным усилиям и последующим переделкам, старайтесь разрешить проблемы на ранних стадиях работы. Далее я покажу несколько далеких от совершенства требований, взятых из реальных проектов. Исследуйте каждое из них, используя перечисленные ранее характеристики качества, и попытайтесь определить проблему. Для начала, например, выясните, поддаются ли они проверке. Если вам не удастся составить тесты, что бы точно сказать, были ли требования реализованы соответствующим образом, возможно, они неясно сформулированы или не хватает необходимой информации.

Я высказал свои соображения о проблемах для каждого требования и предложил несколько путей их решения. Дополнительные проверки еще их улучшат, однако на определенном этапе вы должны приступить к непосредственному написанию ПО. Больше примеров по исправлению неудачных требований изложено Hooks и Farry (2001), Florence i(2002), а также Alexander и Stevens (2002).

Ловушка

Старайтесь избегать паралича аналитического процесса. Нельзя тратить слишком много времени на попытки сделать требования идеальными. Ваша цель — написать требования, которые хороши достаточно, чтобы разработчики могли приступить к конструированию продукта при приемлемом уровне риска.

Пример 1. *«Диспетчер фоновых задач обеспечивает сообщение о состоянии через регулярные интервалы, составляющие не менее 60 секунд».*

Что понимается под сообщениями о состоянии? При каких условиях и как именно они поставляются пользователю? Как долго они остаются видимыми после их отображения? Продолжительность временного интервала не сформулирована точно, а слово «каждый» только вносит дополнительную неясность. Один из способов оценить требование — проверить, устраивают ли пользователя нелепые, но имеющие право на существование интерпретации этого требования. Если нет, то над требованием необходимо еще поработать. В этом примере интервал должен равняться по крайней мере 60 секундам; таким образом, если сообщение будет появляться раз в год, это нормально? А если промежуток не должен превышать 60 секунд, то будет ли интервал, составляющий одну миллисекунду, слишком коротким? Эти утрированные интерпретации не

выходят за рамки первоначального требования, но совершенно очевидно, что пользователь хотел совершенно другого. Из-за этих причин требование нельзя проверить.

А вот возможные способы исправления недостатков этого требования, которыми вы можете воспользоваться после того, как получите дополнительную информацию от клиента.

1. Диспетчер фоновых задач (ДФЗ) отобразит сообщения о состоянии в назначенной области пользовательского интерфейса.

1.1. Сообщения будут обновляться каждые 60 секунд плюс/минус 10 секунд после запуска фоновой задачи.

1.2. Сообщения должны оставаться видимыми постоянно.

1.3. Если взаимодействие с фоновой задачей возможно, то ДФЗ отобразит процент выполнения фоновой задачи.

1.4. По завершению фоновой задачи ДФЗ отобразит сообщение «Выполнено».

1.5. Если выполнение фоновой задачи остановлено, ДФЗ отобразит соответствующее сообщение.

Я разделили это требование на нескольких дочерних, потому что для каждого понадобится отдельный вариант тестирования, кроме того, так каждое проще отслеживать. Если несколько требований объединено в один абзац, то при сборке или тестировании существует опасность пропуска одного из них. В измененном требовании не указан способ отображения сообщения о состоянии. Это проблема проектирования; если вы сейчас определите ее здесь, то разработчики воспримут ее как ограничение, а это может их расстроить. Кроме того, в этом случае вряд ли можно рассчитывать на оптимальный продукт.

Пример 2. *«Редактор XML должен моментально переключаться между режимами отображения и сокрытия непечатаемых символов».*

Компьютеры не могут сделать что-либо моментально, поэтому это требование невыполнимо. Кроме того, это требование неполное, поскольку в нем не указана условие переключения. Выполняет ли это изменение при определенных условиях само ПО или это происходит по инициативе пользователя? Какая часть документа изменяется: выделенный фрагмент текста, весь документ, текущая страница или что-то еще? Какие именно непечатаемые символы: скрытый текст, управляющие символы, тэги разметки или что-то еще? До тех пор пока на эти вопросы не будут получены ответы, требование будет невозможно проверить. Вот как это требование можно улучшить:

«Пользователь сможет переключать отображение и сокрытие всех тэгов XML в редактируемом документе с помощью активации механизма определенного триггера. Отображение должно изменяться в течение 0,1 секунды или менее».

Вот теперь стало ясно, что непечатаемые символы — это тэги разметки XML. Мы знаем, что пользователь инициирует изменение изображения, но требование не ограничивает разработку определением точного механизма. Мы также добавили требование производительности, которое определяет, как быстро отображение должно изменяться. «Моментально» означает «моментально для зрения человека», что вполне достижимо на достаточно быстром компьютере.

Пример 3. *«Анализатор XML выведет отчет об ошибках в разметке, с помощью которого даже пользователи, мало знакомые с языком XML, смогут быстро устранить ошибки».*

Многозначное слово «быстро» относится к действию, выполняемому человеком, а не анализатором. Из-за этого недостаточного объяснения сообщение об ошибке определено не полностью, кроме того, мы не знаем, когда создается этот отчет. Как проверить это требование? Найдите какого-нибудь новичка в XML и посмотрите, сможет ли он быстро исправить ошибки с помощью этого отчета?

Это требование содержит важное понятие определенного класса пользователей — в данном случае это пользователи, мало знакомые с XML, которым нужна помощь ПО для нахождения синтаксических ошибок XML. Аналитик должен найти подходящего представителя этого класса пользователей, чтобы выяснить, какую информацию следует поместить в отчет об ошибке анализатора разметки. Попробуем вместо этого другой способ:

«1. После того, как XML Parser полностью проанализирует файл, он генерирует отчет об ошибках, в котором указан номер строки и текст любых XML-ошибок, обнаруженных в

анализируемом файле, а также описание каждой найденной ошибки.

2. Если никаких ошибок в разметке не было найдено, отчет не генерируется.»

Теперь мы знаем, когда создается отчет об ошибках и что в нем содержится, однако то, как отчет будет выглядеть, мы оставили на усмотрение проектировщика. Мы также сформулировали условие исключения, которое первоначальное требование не затрагивает: если ошибки he найдено, генерировать отчет не надо.

Пример 4, *«Если возможно, номера счетов следовало бы проверить, по списку корпоративных счетов».*

Что означает «если возможно»? Технически осуществимо? Или когда доступен основной список счетов во время запуска? Если вы не уверены, можно ли реализовать функцию, сделайте пометку «TBD», чтобы указать, что эта проблема еще не решена. После проверки одно из двух будет ликвидировано — либо пометка «TBD», либо требование. В требовании не указано, что произойдет, если проверка пройдет успешно или окончится неудачей. Избегайте неточных слов вроде «следовало бы». Некоторые авторы требований пытаются выразить тонкие различия, используя «будет», «следовало бы» и «возможно», чтобы подчеркнуть значимость. Я предпочитаю использовать «будет» или «должен» как ясное указание на цель требования и приоритеты. Вот как выглядит исправленный вариант требования.

«В момент ввода номера счета система проверит его по основному корпоративному списку счетов. Если номер счета в списке не найден, система отобразит сообщение об ошибке и не примет заказ».

Связанное требование будет относиться к условию исключения: основной список счетов не доступен во время проверки достоверности.

Пример 5. *«В редакторе не будет функций поиска и замены, результаты которых могут быть катастрофичными».*

Понятие «катастрофичные результаты» можно интерпретировать по-разному. Непреднамеренное глобальное изменение может обернуться катастрофой, если пользователь не обнаружит ошибку или не может ее исправить. Будьте благоразумны при составлении обратных требований, описывающих, что система не будет делать. Основная забота в этом примере — защита содержимого файла от непреднамеренных повреждений или потерь. Возможно, настоящие требования выглядят вот так.

«1. Редактор будет запрашивать подтверждения от пользователя, где тот должен подтвердить глобальные изменения текста, удаления и вставки, которые могут привести к потере данных.

2. В приложении будет реализована многоуровневая функция отмены, ограниченная только системными ресурсами, которые доступны приложению.»

Пример 6. *«Испытательный прибор позволит пользователю легко подключить дополнительные компоненты, в том числе импульсный генератор, вольтметр, измеритель емкости и нестандартные зондовые платы».*

Это требования к продукту, содержащему встроенное ПО, которое используется для тестирования различных типов измерительных приборов. Слово «легко» подразумевает требование легкости и простоты использования, но оно не измеримо и не поддается проверке. «В том числе» не дает ясности, полный ли это список внешних устройств, которые должны быть подключены к испытательному устройству, или существует еще множество других приборов, о которых мы не знаем. Вот какие альтернативные требования содержат хорошо обдуманные ограничения дизайна.

«1. Испытательное устройство содержит USB-порт, чтобы пользователь смог подключить любое измерительный прибор, у которого есть USB-подключение.

2. USB-порт будет установлен на передней панели для того, чтобы позволить квалифицированному оператору подключить измерительный прибор за 15 секунд или менее.»

Словарь данных

Давным-давно я вел проект, в котором три программиста иногда небрежно использовали различные имена, длину переменных и критерий достоверности для одного и того же элемента. Результат очевиден: путаница с настоящим определением, повреждение данных и головная боль при

обслуживании. Мы пострадали из-за отсутствия ***словаря данных*** (data dictionary) — общего хранилища, где определены значение, тип данных, длина, формат, необходимая степень точности и допустимые диапазоны значений или список значений всех элементов данных и атрибутов, используемых в приложении.

Информация в словаре данных связывает различные представления требований. Проблема целостности становится меньше, если все разработчики будут придерживаться определений из словаря данных. Начните собирать определения данных еще в процессе выявления требований. Словарь данных дополнит словарь проекта, где приведены определения отдельных терминов. Вы можете объединить эти документы, хотя лично я предпочитаю хранить их отдельно. Словарем данных можно управлять, как приложением к спецификации требований к ПО или как отдельным документом или файлом.

По сравнению с определениями данных, размещенными в различных местах функциональных требований, отдельный словарь данных облегчает поиск необходимой информации, а также помогает избежать ненужных повторов и несогласованности. В некоторые коммерческие средства анализа и проектирования входит компонент словаря данных. Если вы устанавливаете словарь данных вручную, подумайте о применении гиперссылок. Щелкнув элемент данных, который представляет собой часть определения структуры данных, вы перейдете к определению этого элемента данных; таким образом, перемещаться по иерархическому дереву определений легко и просто.

Элементы в словаре данных представлены с помощью простой нотации (DeMarco, 1979; Robertson и Robertson, 1994). Определяемый элемент расположен слева от знака равенства, а его определение — справа. Так определяются простейшие элементы данных, объединение нескольких элементов данных в структуры, необязательные элементы данных, итерация (повтор) элемента данных и список значений элемента данных. Следующие примеры взяты из проекта Chemical Tracking System (ну откуда же еще!).

Простейшие элементы данных. Простейшим элементом данных называется тот, дальнейшее упрощение которого невозможно или ненужно. Его определение содержит тип простейших данных, размер, диапазон допустимых значений и другие подобные атрибуты. Простейшие данные определяются с помощью комментария, т.е. текста, ограниченного звездочками:

*Request ID = * 6-значный, сгенерированный системой, последовательный номер, состоящий из целого числа, начинающийся с 1, уникальным образом идентифицирующий каждый запрос **

Структура. Структура данных или записей содержит несколько элементов данных. Если элемент в структуре данных является необязательным, поставьте его в скобки:

Запрошенный химикат=ID химиката
+ Количество контейнеров
+ Класс
+ Объем
+ Единицы объема
+ (Поставщик)

Эта структура определяет всю информацию, связанную с запросом определенного химиката. ***Поставщик*** (vendor) является необязательным элементом, поскольку сотруднику, разместившему запрос, может быть безразлично, кто именно поставил химикат. Каждый элемент данных в структуре должен быть определен в словаре данных. Структуры могут содержать другие структуры.

Повтор. Если в структуре данных содержится несколько экземпляров элемента данных, заключите этот элемент в фигурные скобки. Покажите допустимое количество возможных повторов в формате ***минимум:максимум*** перед открывающей скобкой:

Запрос=ID Запроса
+ Дата запроса
+ Номер счета
+1:10(Запрошенный химикат)

Этот пример показывает, что запрос химиката должен содержать по крайней мере один, но не более 10 химикатов. К каждому запросу добавлены атрибуты идентификатора, дата, когда запрос был создан, и номер счета для оплаты.

Выбор. Элемент данных, который может принять ограниченное число отдельных значений, называется ***перечисляемым простейшим.*** Покажите возможные значения в списке внутри квадратных скобок, где элементы списка разделены вертикальными разделителями:

Единицы количества = [«граммы» | «килограммы» | «миллиграммы» | «каждый»]

* текстовая строка, состоящая из 9 символов, указывающая

единицы, связанные с количеством запрошенного химиката *

Эта запись указывает на то, что существует только четыре допустимых значения для текстовой строки ***Единицы количества***. Комментарий, ограниченный звездочками, — это текстовое определение элемента данных.

Время, потраченное на создание словаря данных, будет более чем компенсировано временем, которые вы сэкономите, избежав ошибок, возможных, если участники проекта по-разному понимают ключевые данные. Если вы регулярно обновляете словарь данных, он останется ценным средством и при обслуживании системы, и при разработке схожих систем.

Что теперь?

- Проверьте страницу функциональных требований в спецификации требований к ПО вашего проекта и убедитесь, что каждое положение демонстрирует характеристики отличных требований. Перепишите все требования, которые не отвечают этому уровню.
- Если в вашей организации еще нет стандартного шаблона спецификации требований к ПО, соберите небольшую рабочую группу для его обсуждения. Начните с шаблона на рис. 10-1 и измените его так, чтоб он наилучшим образом соответствовал потребностям ваших проектов и продуктов. Выработайте стандарт именования отдельных требований.
- Пригласите от трех до шести лиц, заинтересованных в проекте, для проверки спецификации требований к ПО для вашего проекта (Wiegers, 2002a). Убедитесь, что каждое требование отвечает соответствующим характеристикам, о которых говорилось в главе 1. Проверьте спецификацию на наличие конфликтов между различными требованиями в спецификации, недостающих требований и недостающих разделов. Убедитесь, что обнаруженные недостатки, исправлены в спецификации требований к ПО и в любых предыдущих продуктах, созданных на основе этих требований.
- Возьмите комплексный объект данных из одного из ваших приложений и определите его с помощью образца словаря данных, представленного в этой главе.

Глава 11 Любое изображение стоит 1024 слов

Специалисты, работающие над проектом Chemical Tracking System, выполняли первую проверку спецификации требований к ПО. В ней участвовали Дейв (менеджер проекта), Лори (аналитик требований), Элен (ведущий разработчик), Рамеш (руководитель тестирования), Тим (сторонник продукта от химиков) и Роксана (апологет продукта от специалистов, работающих на складе химикатов). Тим начал со слов: «Я прочитал всю спецификацию требований. Большинство требований мне показались нормальными, но некоторые дались с трудом. Я не уверен, что мы определили все этапы для процесса запроса химиката».

«Мне было трудно продумать все варианты тестирования для того, чтобы раскрыть все возможные изменения статуса запроса, — добавил Рамеш. — Я обнаружил, что множество требований, касающихся изменения статуса, разбросаны по всей спецификации, но не могу твердо сказать, пропущены ли какие-то из них. Мне показалось, что пара требований конфликтует». Роксана обнаружила похожую проблему. «Меня сбило с толку описание того, как непосредственно выполнять запрос химиката, — сказала она. — Отдельные функциональные требования имели смысл, но мне трудно представить всю последовательность действий при запросе».

После того как остальные участники поделились своими опасениями, Лори подвела итоги: «Похоже, с помощью спецификации нам не удастся узнать о системе все, что необходимо. Я нарисую несколько картинок, чтобы нам было легче представить требования и понять, не прояснит ли это проблемные области. Спасибо за то, что высказали ваше мнение».

Согласно авторитетному мнению специалиста в области требований Alan Davis, никакое представление требований в одном виде не дает их полной картины (Davis, 1995). Необходима комбинация текстовых и визуальных способов представления требований, чтобы получилась полная картина создаваемой системы. К таким способам относятся списки и таблицы, графические модели анализа, прототипы пользовательского интерфейса, варианты тестирования, деревья решений и таблицы решений. В идеале различные представления требований должны создавать разные специалисты. Аналитик может написать функциональные требования и нарисовать отдельные модели, дизайнер пользовательского интерфейса — создать прототип, а руководитель тестирования — написать варианты тестирования. Сравнение представлений, созданных различными специалистами в ходе разнообразных исследований, помогает выявить несоответствия, неясности, допущения и упущения, которые трудно обнаружить, когда требования представлены в одном формате.

Определенные типы информации с помощью диаграмм удается связать гораздо эффективнее, чем с помощью текста. Изображения помогают преодолеть языковые барьеры и терминологические разногласия между членами команды. Аналитику придется разъяснить остальным заинтересованным лицам назначение используемых моделей и применяемые условные обозначения. В этой главе рассказывается о нескольких приемах моделирования требований с иллюстрациями и ссылками на другие источники, где можно получить более подробную информацию.

Моделирование требований

Когда много лет назад я начал рисовать модели анализа, я надеялся найти единственный прием, позволяющий собирать всю информацию в одно целостное отображение требований к системе. Со временем я понял, что подобной всеобъемлющей диаграммы не существует. Первоначальной целью структурированных систем анализа была полная замена классической функциональной спецификации графическими диаграммами и системами обозначений, более формальными, чем текстовые комментарии (DeMarco, 1979). Однако опыт показал, что модели анализа должны дополнять — а не заменять — спецификации требований на естественном языке (Davis, 1995).

К моделям визуального представления относятся ***диаграммы потоков данных*** (data flow diagrams, DFD), ***диаграммы «сущность - связь»*** (entity-relationship diagrams, ERD), ***диаграммы***

перехода состояний .(state-transition diagrams, STD), называемые также ***диаграммами состояний, карты диалогов*** (dialog maps), ***диаграммы вариантов использования*** (о которых рассказывалось в главе 8) диаграммы классов и диаграммы взаимодействия (о них также рассказывалось в главе 8). Системы обозначений, представленные здесь, обеспечивают общий, стандартный для всей индустрии язык, которые пользуются участники проекта. Конечно, вы можете специально разработать диаграммы, чтобы дополнить возможности устного и письменного общения в рамках проекта, однако не факт, что пользователи интерпретируют их одинаково. В то же время нельзя не признать ценность нестандартных приемов моделирования. Одна команда применяла средство для составления графика моделирования временных требований для встроенного ПО, причем за единицу измерения были приняты миллисекунды, а не дни или недели.

Эти модели годятся для разработки и изучения требований, а также для построения ПО. Будете ли вы использовать их для анализа или для дизайна, зависит от временных рамок и целей моделирования. Применение этих диаграмм для анализа требований позволит вам смоделировать предметную область или создать концептуальные представления новой системы. Они отображают логические аспекты компонентов данных предметной области, транзакции и преобразования, объекты реального мира и изменения состояния системы. Вы можете создавать модели на основании текстовых требований, чтобы отобразить их с различных точек зрения, или вывести детализированные функциональные требования из моделей высокого уровня, созданных на основе предоставленной пользователями информации. В процессе разработки модели демонстрируют, как вы намерены реализовать систему: ту базу данных, которую вы планируете создать, классы объектов, которые вы будете иллюстрировать примерами, и модули кода, которые вы собираетесь разрабатывать.

Ловушка

Не следует полагать, что разработчики смогут просто преобразовать модели анализа в код, не разобрав подробно весь процесс. Поскольку в обоих типах диаграмм используются одни и те же системы обозначений, ясно назовите каждый из них моделью анализа (концепция) или моделью проектирования (то, что вы планируете создать).

Приемы моделирования анализа, описанные в этой главе, поддерживаются различными ***коммерческими инструментами автоматизированного проектирования ПО*** (computer-aided software engineering, CASE). Использование этих инструментов обеспечивает некоторое .преимущество перед обычными средствами рисования. Во-первых, они легко позволяют улучшить качество диаграмм при повторных просмотрах требований. Вам не удастся создать отличную модель с первого раза, поэтому итерацию можно назвать ключом к успеху при моделировании систем (Wiegers, 1996a). Кроме того, инструменты автоматизированного проектирования ПО «знают» правила для каждого метода моделирования, который они поддерживают. Они способны определить синтаксические ошибки и несоответствия, которые специалистам, проверяющим диаграммы, не всегда удается обнаружить. Эти инструменты связывают различные диаграммы друг с другом и с их общими определениями данных в словаре данных, а также позволяют поддерживать модели в согласованном состоянии и в соответствии с функциональным требованиям в спецификации требований к ПО.

Редкий случай, когда команде необходимо создать полный набор моделей анализа для всей системы. При моделировании сосредоточьтесь на наиболее сложных и опасных участках системы, а также на те, где наиболее вероятны неясности и неопределенности. Элементы системы, от которых зависит ее безопасность и защита, а также элементы, влияющие на выполнение критически важных задач, можно смело назвать кандидатами на моделирование, так как последствие дефектов в них окажутся особенно тяжелыми.

От желания клиента к модели анализа

Тщательно слушая, как клиенты представляют свои требования, аналитик может выделить

ключевые слова, которые преобразуются в определенные элементы модели. В табл. 11-1 перечислены возможное отображение значимых существительных и глаголов, употребляемых пользователями, в компонентах модели, о которых рассказано далее в этой главе. По мере того как вы будете записывать пожелания пользователей в виде требований и моделей, вам придется связать каждый компонент модели с определенным пользовательским требованием

Таблица 11-1. Привязка пожеланий клиента к компонентам модели анализа

Тип слова	Примеры	Компоненты модели анализа
Существительные	Люди, организации, системы ПО, элементы данных или существующие объекты	• Оконечные объекты или хранилища данных (диаграммы потоков данных) • Действующие лица (диаграммы вариантов использования) • Объекты или их атрибуты (диаграммы «сущность - связь») • Классы или их атрибуты (диаграммы классов)
Глаголы	Действия, задачи, которые пользователь может выполнить, или события, которые могут произойти	• Процессы (диаграммы потоков данных) • Варианты использования (диаграммы вариантов использования) • Взаимосвязи (диаграммы, «сущность - связь») • Преобразования (диаграммы перехода состояний) • Процессы (диаграммы процессов)

В этой книге я использовал в качестве учебного примера Chemical Tracking System. В следующем абзаце для этой системы описаны потребности пользователей, который представил сторонник продукта от класса Химики. Значимые существительные выделены **полужирным** начертанием, а глаголы или отглагольные существительные — *курсивом;* отыщите эти ключевые слова в моделях анализа, показанных далее в этой главе. На схемах некоторых моделей с целью иллюстрации показана информация, выходящая за рамки содержания следующего абзаца, тогда как на других моделях отображена только часть информации, представленной здесь:

Химик или кто-то из **работающих на складе химикатов** может *разместить* **запрос** на один или несколько **химикатов.** Запрос может быть *выполнен* или посредством *доставки* **контейнера** с химикатом, который числится **инвентарной описи товаров на складе химикатов,** или посредством *размещения* заказа на химикат у постороннего **поставщика. Сотрудник,** размещающий запрос, при *подготовке* запроса должен иметь возможность *искать* в режиме реального времени нужный химикат в **каталогах поставщиков.** Системе необходимо *отслеживать* **состояние** каждого запроса с момента его подготовки и до момента его выполнения или *отмены*. Ей также необходимо отслеживать историю каждого контейнера с химикатом с момента его *получения* компанией до его полного *расходования* или *уничтожения*».

Диаграмма потока данных

Диаграмма потока данных (data flow diagram, DFD) — основной инструмент структурного анализа (DeMarco, 1979; Robertson и Robertson,. 1994). Она позволяет определять трансформационные процессы системы, совокупность (хранение) данных или материалов, которыми система управляет, и потоки данных или материалов между процессами, хранилищами и внешним миром. При моделировании потоков данных для системного анализа используется прием разложения функций, при котором сложные проблемы раскладываются на составляющие, размещаемые по нарастающей по уровням детализации. Этот метод отлично подходит для систем обработки транзакций и других приложений с большим набором функций. Посредством добавления элементов управления потоком диаграмму потока данных удалось применить для моделирования систем, работающих в режиме реального времени (Hatley, Hruschka и Pirbhai, 2000).

Диаграмма потока данных позволяет представить этапы бизнес-процесса или операции предложенной системы ПО. Я часто пользовался этим средством, опрашивая клиентов — я набрасывал на доске диаграмму, пока длилось обсуждение бизнес-операций. Диаграммы потока данных могут представлять системы на самых разных уровнях абстракции: диаграммы высокого уровня предоставляют целостный, панорамный вид данных и выполняемых компонентов в многоэтапном процессе, дополняющем точное, детальное представление, реализуемое в спецификации требований. Диаграмма потока данных иллюстрирует то, как совмещаются функциональные требования в спецификации, чтобы пользователь смог выполнять определенные задачи, например запрос химиката.

Ловушка

Не обольщайтесь, что клиентам заранее известно, как читать модели анализа, однако не думайте, что они не смогут разобраться в них. Объясните цели и условные обозначения для каждой модели сторонникам продукта и попросите их просмотреть диаграммы.

Контекстная диаграмма на рис. 5-3 — это наивысший уровень абстракции диаграммы потока данных. Контекстная диаграмма представляет всю систему как единый процесс, изображенный в виде круга (*пузырька).* На ней также показаны ***оконечные объекты*** (terminators), или внешние объекты, подсоединенные к системе и данным или материальным потокам между системами и оконечными объектами. Потоки на контекстной диаграмме часто представляют сложные структуры данных, определенные в словаре данных.

Вы можете конкретизировать контекстную диаграмму до уровня 0, выделив важнейшие процессы системы. На рис. 11-1 показан уровень 0 диаграммы потока данных для Chemical Tracking System (несколько упрощенный). Единый процесс Chemical Tracking System, обозначаемый на контекстной диаграмме одним кружком, был разделен на семь основных процессов (кружков). Как и на контекстной диаграмме, оконечные объекты показаны прямоугольниками. Все потоки данных (стрелки), присутствовавшие на контекстной диаграмме, отражены на уровне 0 диаграммы потока данных. Кроме того, парами параллельных линий здесь обозначены *хранилища данных,* которые относятся ко внутренней части системы и, следовательно, не показаны на контекстной диаграмме. Стрелка от кружка к хранилищу, указывает, что данные были помещены в хранилище, стрелка, выходящая из хранилища, обозначает операцию считывания, а двунаправленная стрелка между хранилищем и кружком — операцию обновления.

Каждый процесс, изображенный в виде отдельного кружка на уровне 0 диаграммы, можно расписать более подробно с помощью отдельной диаграммы, изобразив на ней всю функциональность этого процесса. Аналитик продолжает последовательные уточнения до тех пор, пока диаграммы нижних уровней не будут содержать только простейшие операции, которые можно ясно представить в виде комментариев, псевдокода, графика или диаграммы процесса. Функциональные требования в спецификации требований к ПО точно определят, что происходит в ходе каждого простейшего процесса. Диаграммы потока данных каждого уровня должны быть сбалансированы и согласованы с расположенным выше уровнем так, чтобы вся входящие и исходящие потоки на дочерней диаграмме соответствовали аналогичным на родительской диаграмме. Сложные потоки данных в диаграммах высоких уровней могут быть разделены на составные элементы, как определено в словаре данных, на диаграммах потока данных нижних уровней.

На первый взгляд схема на рис.11-1 выглядит сложной. Однако если вы изучите ближайшее окружение любого процесса, то увидите элементы данных, которые он принимает и отдает, а также их исходные точки и точки назначения. Чтобы увидеть, как именно в процессе используются элементы данных, вам понадобится либо нарисовать более подробную дочернюю диаграмму потока данных или сослаться на функциональные требования для этой части системы.

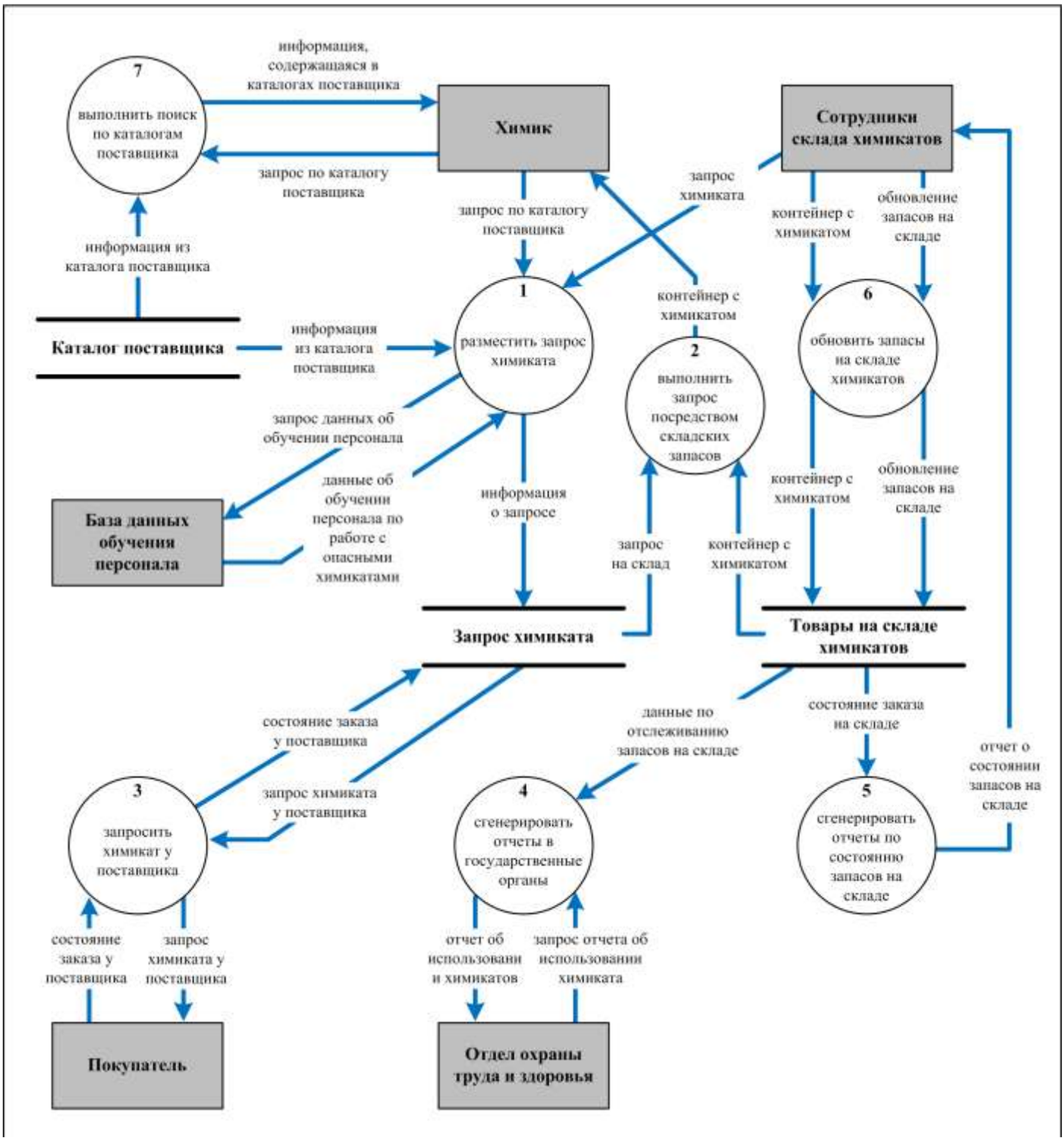


Рис. 11-1. Уровень 0 диаграммы потока данных для Chemical Tracking System

Ниже приведены некоторые правила для рисования диаграмм потока данных. Не все придерживаются одних и тех же правил (например, некоторые аналитики показывают оконечные объекты только на контекстных диаграммах), однако я считаю их полезными. Использование моделей для улучшения взаимодействия участников проекта важнее догматического соблюдения этих правил.

- размещайте хранилища данных только на уровне 0 и более низких уровнях диаграммы потока данных, а не на контекстной диаграмме;
- процессы взаимодействуют через хранилища данных, а не посредством прямых потоков от одного процесса к другому. Точно так же данные не могут передаваться напрямую из одного хранилища в другое, они должны пройти через процесс (кружок);

- не пытайтесь вообразить последовательность этапов процесса с помощью диаграммы потока данных; называйте каждый процесс как короткое действие: глагол + объект (например, создать инвентарный отчет). Используйте названия, понятные клиентам и употребляемые в деловой или предметной области;
- присваивайте процессам уникальные номера согласно иерархии. На уровне 0 диаграммы, пронумеруйте каждый процесс, используя целые числа. Если вы создадите дочернюю диаграмму потока данных для процесса 3, пронумеруйте процессы на ней как 3.1, 3.2 и т.д.;
- не показывайте на одной диаграмме более 8-10 процессов, в противном случае ее будет трудно рисовать, изменять и понимать. Если у вас больше десяти процессов, создайте еще один уровень абстракции, сгруппировав связанные процессы в процесс более высокого уровня;
- кружки с только входящими или только исходящими потоками вызывают подозрение. Корректный процесс, изображаемый кружком на диаграмме потока данных, как правило, отличает наличие и входящих, и исходящих потоков.

Когда представители клиентов просматривают диаграмму потока данных, они должны убедиться, что на ней отображены все известные процессы и что у них нет отсутствующих или ненужных входящих или исходящих потоков. При изучении диаграммы потока данных часто выявляются не распознанные ранее классы пользователей, бизнес-процессы и соединения с другими системами.

Диаграмма «сущность - связь»

Точно так же, как диаграмма потока данных иллюстрирует процессы, происходящие в системе, модель данных отображает взаимоотношения данных. Широко используется такая модель данных, как ***диаграмма «сущность - связь»*** (entity-relationship diagrams, ERD) (Wieringa,1996). Если диаграмма «сущность — связь» представляет логические группы информации предметной области и их взаимосвязи, вы используете диаграмму «сущность - связь» в качестве инструмента анализа требований. Анализ диаграммы «сущность - связь» понять и связать компоненты данных компании или системы, не подразумевая, что в продукт будет обязательно включена база данных. И наоборот, когда вы создаете эту диаграмму в ходе разработки системы, вы определяете логическую или физическую структуру базы данных системы.

Объектами (entities) называются физические элементы (включая людей) или агрегация элементов данных, важных для бизнеса, анализ которого вы выполняете, или для системы, которую вы планируете создать (Robertso и Robertson, 1994). Объекты называются посредством существительных в единственном числе, они показаны в прямоугольниках на диаграмме «сущность - связь». На Рис. 11-2 показана часть диаграммы «сущность - связь» для Chemical Tracking System с использованием одной из нескольких применяемых для диаграмм такого типа систем обозначений. Обратите внимание, что объекты «Запрос химиката», «Каталог поставщика» и «Товары на складе химикатов» на диаграмме потока данных на рис. 11-1 показаны в виде хранилища данных. Другие объекты представляют действующие лица, взаимодействующие с системой («Сотрудник, разместивший заказ на химикат»), физические элементы, являющиеся частью деловых операций («Контейнер с химикатом»), и блоки данных, которые не показаны на уровне 0 диаграммы потока данных, но показаны на ее более низком уровне («История контейнера», «Химикат»).

Каждый объект описывается несколькими атрибутами; у отдельных экземпляров объекта будут различные значения атрибутов. Например, в атрибуты каждого химиката включены уникальный химический идентификатор, строгое название химиката и графическое представление его химической структуры. В словаре данных приводятся детальные определения этих атрибутов — это гарантирует, что объекты на диаграмме «сущность - связь» и соответствующие им хранилища данных на диаграмме потока данных определены идентично.

Ромбы на диаграмме «сущность – связь» обозначают ***взаимоотношения*** (relationships), показывающие логические и числовую связь пар объектов. Названия взаимоотношениям даются в соответствии с природой соединений. Например, взаимоотношения между элементами «Сотрудник, разместивший заказ на химикат», и «Запросом химиката» определяются *как размещение.* Вы можете

прочитать это взаимоотношения как «Сотрудник размещает Запрос на химикат» или как «Запрос на химикат размещен Сотрудником». Согласно некоторым правилам, фигуру в форме ромба следует назвать «размещен тем-то», что имеет смысл, только если читать диаграмму слева направо. Когда вы покажете клиентам диаграмму «сущность - связь», попросите их проверить, все ли взаимоотношения показаны корректно и соответствующим образом. Также попросите их определите все возможные взаимоотношения с объектами, которые не показаны на модели.

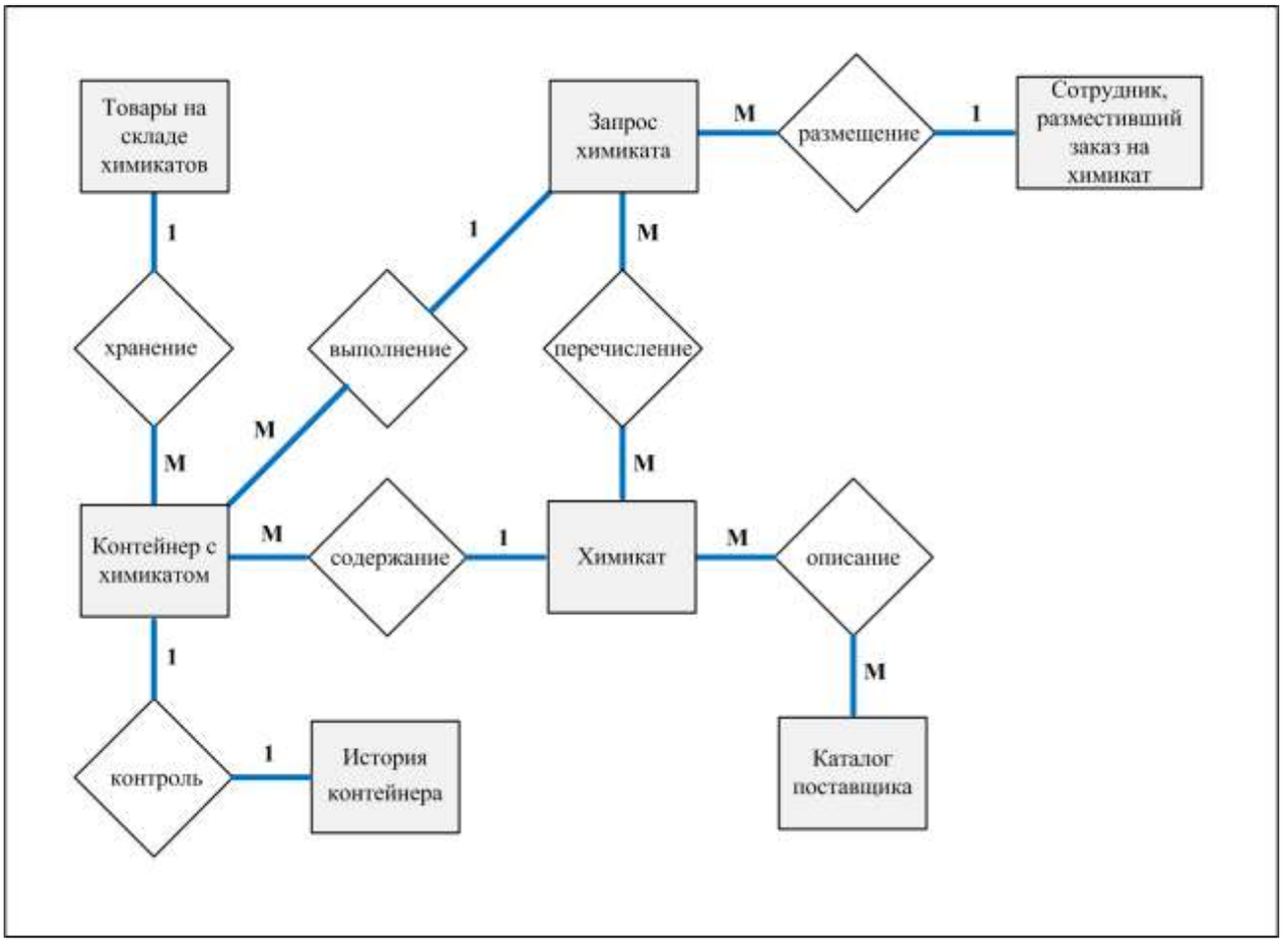


Рис. 11-2. Часть диаграммы «сущность - связь» для Chemical Tracking System

Количество элементов (cardinality) каждого взаимоотношения, или его множественность, показаны цифрой или буквой на прямых, соединяющих объекты и взаимоотношения. Для диаграмм «сущность – связь» используются различные правила для обозначения количества элементов; условные обозначения на рис. 11-2 используются наиболее часто. Поскольку каждый Сотрудник, разместивший заказ на химикат, может разместить несколько запросов, то между элементами «Сотрудник, разместивший заказ на химикат», и «Запрос химиката» существует связь «один ко многим». Количество элементов показано цифрой ***1*** на линии, соединяющей элемент «Сотрудник, разместивший заказ на химикат», и взаимоотношение *«размещение»,* и буквой ***M*** (много) на линии, соединяющей элемент «Запрос химиката» и взаимоотношение *«размещение».* Ниже перечислены другие возможные случаи:

- один к одному (каждый контейнер с химикатом отслеживается в одной Истории контейнера);
- многие ко многим (в каждом каталоге поставщика содержится множество Химикатов, а некоторые Химикаты встречаются в нескольких Каталогах поставщика).

Если вам известно более точное количество элементов, чем просто *много,* вы можете указать вместо общего M конкретное число или диапазон.

Моделирование проблем, а не ПО

Как-то я в качестве представителя специалистов по ИТ работал в команде, которая занималась модернизацией бизнес-процессов. Наша задача заключалась в снижении в 10 раз времени, необходимого для того, чтобы новый химикат оказывался доступным для использования в продукте. В команду были включены представители следующих подразделений компании, задействованных в коммерциализации химиката:

- химиков-синтетиков, которые создали новое вещество;
- химиков-технологов, разрабатывающих процесс промышленного производства вещества;
- химиков-аналитиков, разрабатывающих методы проверки чистоты вещества;
- патентных поверенных, которые занимаются патентной защитой изобретения;
- специалистов отдела охраны труда и здоровья, получающих разрешение правительства на использование химиката в потребительских товарах.

После того как мы разработали новый процесс, который по нашему мнению должен был необыкновенно ускорить коммерческое применение химиката, я опросил членов команды, ответственных за каждый этап процесса. Я задал каждому два вопроса: «Какая информация вам нужна для выполнения этого этапа?» и «Какую информацию, которая появится в результате выполнения этого этапа, следует сохранить?». Собрав и сравнив ответы, я выявил те этапы, информацию для которых не поставлял никто, а также этапы, выходная информация которых не была нужна никому. Мы разрешили эти проблемы.

Затем я нарисовал диаграмму потока данных, чтобы проиллюстрировать новый процесс коммерциализации химиката и диаграмму «сущность - связь» для моделирования взаимоотношений данных. Все наши элементы данных были определены в словаре. Такие модели анализа будут полезны в качестве инструментов взаимодействия, которые помогут членам команды выработать общее понимание нового процесса. Кроме того, модели представляют собой прекрасную точку отсчета для разработки требований для приложений и определения их границ.

Диаграмма перехода состояний

Для любых систем ПО необходимо учитывать комбинацию функционального поведения, особенности управления данными и изменения состояния. Состояний, в которых в каждый момент времени могут находиться системы реального времени и приложения, управляющие процессами, немного. Изменение состояния происходит, только когда удовлетворяются четко определенные критерии, такие, как получение определенного сигнала на входе при определенных условиях. Примером может служить перекресток магистрали с установленными датчиками движения, защищенной полосой поворота, а также разметкой и сигналами для пешеходного перехода. Многие информационные системы имеют дело с бизнес-объектами такими как заказ на покупку, счета-фактуры, товарно-материальные ценности и т.п., для жизненных циклов которых возможно несколько различных состояний. Элементы системы, для которых возможен набор состояний и их

изменение, называются ***механизмами с конечным числом состояний*** (finite-state machines) (Booch, Rumbaugh, Jacobson, 1999).

При описании сложного механизма с конечным числом состояний на естественном языке высока вероятность упустить из внимания разрешенное изменение состояния или появления запрещенного изменения. В зависимости от структуры спецификации требований к ПО, требования, относящиеся к поведению состояния механизма, могут быть в ней разобщены. В этом случае весьма трудно понять поведение системы.

Диаграмма перехода состояний (state-transition diagram) позволяет получить точное, полное и ясное представление о механизме с конечным числом состояний. Связанный с этой моделью прием — *диаграмма состояния* — включен в обладающий в некотором смысле более богатым набором условных обозначений ***унифицированный язык моделирования*** (Unified Modeling Language, UML), который моделирует состояния объекта в течение его жизненного цикла (Booch, Rumbaugh, Jacobson, 1999). Диаграмма перехода состояний содержит три типа элементов:

- возможные состояния системы — показаны в виде прямоугольников;
- разрешенные состоянием ***переходы*** (transitions), — показаны в виде стрелок, соединяющих пары прямоугольников;
- события или условия, вызывающие каждый переход, — показаны в виде текстовых пояснений для каждой стрелки перехода. Текст может пояснять и событие, и соответствующую реакцию системы.

На рис. 11-3 показана часть диаграммы перехода состояний для домашней системы безопасности. В диаграмме перехода состояний для системы реального времени предусмотрено особое состояние, обычно называемое Ожидание (эквивалентно состоянию Отключено на рисунке), в которое система возвращается, когда она не занята другими процессами. В отличие от этого, диаграмма перехода состояний для объекта, имеющего определенный жизненный цикл, например Запрос химиката, имеет одно или более конечных состояний, которые означают финальные значения состояния объекта.

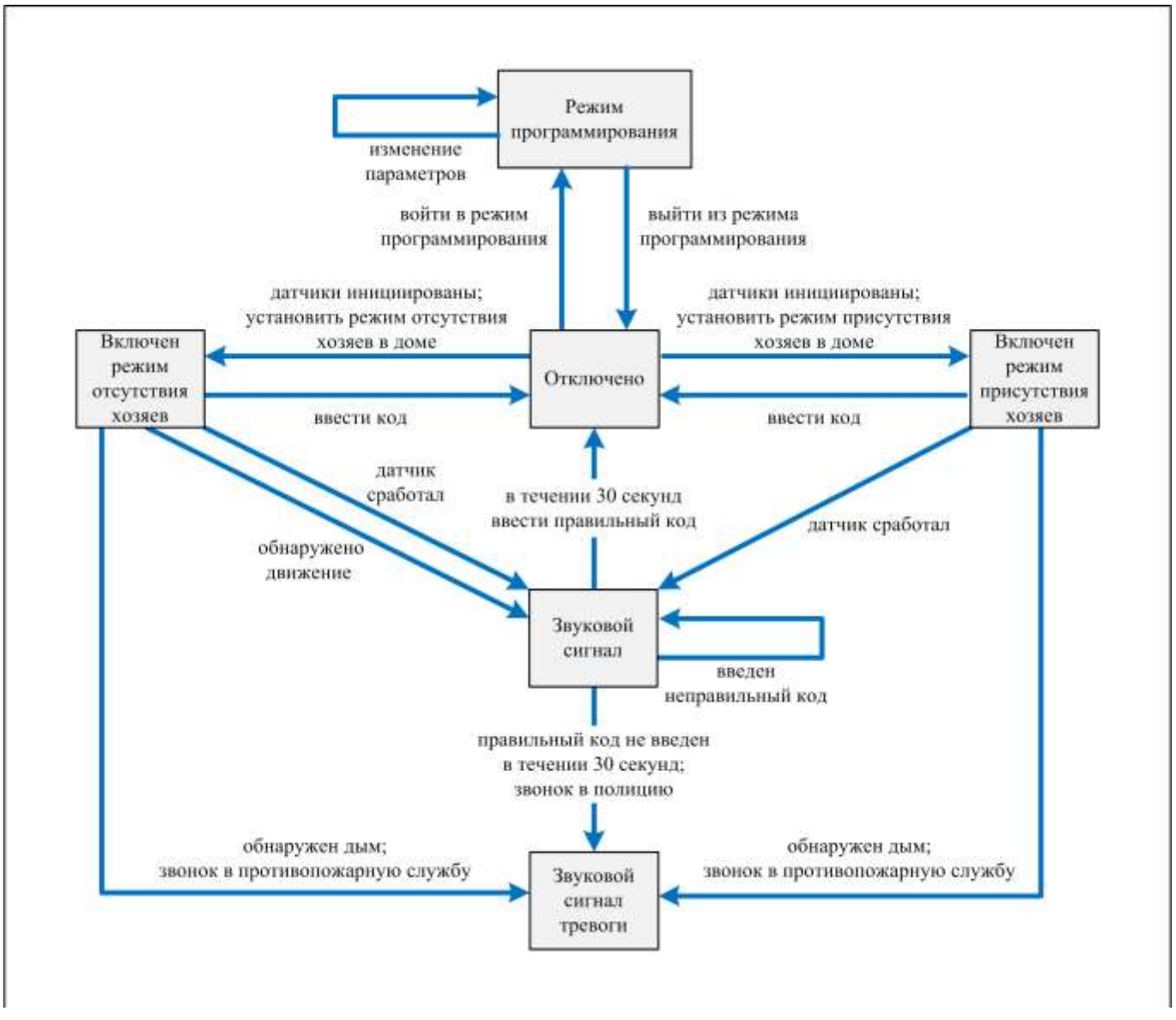


Рис. 11-3. Часть диаграммы перехода состояний для домашней системы безопасности

На диаграмме перехода состояний не показаны детали процессов, выполняемых системой, а только возможные изменения состояний, возникающие в результате этих процессов. Диаграмма перехода состояний помогает разработчику понять предполагаемое поведение системы. Это хороший метод проверки того, все ли необходимые состояния и переходы состояний корректно и полно описаны в функциональных требованиях. Тестировщики могут вывести варианты тестирования на основании диаграммы перехода состояний, в которую включены все допустимые пути переходов. Клиентам, как правило, хватает незначительных пояснений, касающихся принятой системы обозначений, — что означают прямоугольники и стрелки, — чтобы прочитать диаграмму перехода состояний.

Вспомните из главы 8, что основная функция Chemical Tracking System — позволить действующим лицам, которые названы «Сотрудники, разместившие заказ на химикат», разместить запрос химиката, который может быть выполнен либо складскими работниками (если химикат есть на складе), либо сторонним поставщиком (для этого ему надо отправить запрос). Каждый запрос проходит через несколько состояний с момента его создания до момента либо его выполнения, либо отмены (два конечных состояния). Таким образом, мы можем обрабатывать жизненный цикл запроса химиката как механизм с конечным числом состояний и моделировать его так, как показано на рис. 11-4.

Эта диаграмма перехода состояний показывает, что отдельный запрос может находиться в одном из следующих семи состояний:

- **В подготовке.** Сотрудник создает новый запрос, инициировав эту функцию из другой части системы;
- **Отложен.** Сотрудник сохраняет часть запроса для выполнения его в будущем, не передавая его в систему и не отменяя операцию запроса;
- **Принят.** Пользователь отправляет законченный запрос химиката, и система принимает его к исполнению;
- **Размещен.** Запрос должен быть удовлетворен сторонним поставщиком, покупатель размещает заказ у продавца;
- **Выполнен.** Запрос удовлетворен: контейнер с химикатом поставлен либо со склада химикатов, либо от поставщика;
- **Заказ ожидает выполнения.** У продавца не оказалось химиката в наличии, и он уведомил покупателя, что заказ отложен для будущей поставки;
- **Отменен.** Сотрудник отменил принятый системой заказ до того, как тот был выполнен, или покупатель отменил заказ у продавца, до того как тот был выполнен или пока он был отложен.

Когда представители пользователей Chemical Tracking System просмотрели диаграмму перехода состояний для запроса химиката, они определили, что одно состояние не нужно, другое важное состояние отсутствует, и указали два неправильных перехода. При изучении соответствующих функциональных требований эти ошибки никто из них не заметил. В этом и заключается важность представления информации о требованиях на нескольких уровнях абстракции. Зачастую проблему легче обнаружить, если идти назад от наиболее детализированного уровня и рассматривать большую картину, которую обеспечивают модели анализа. Однако диаграмма перехода состояний не дает уровень деталей, достаточный разработчику для создания ПО. Следовательно, в спецификацию требований к ПО для Chemical Tracking System были включены функциональные требования, связанные с обработкой запроса химиката и возможными изменениями его состояния.

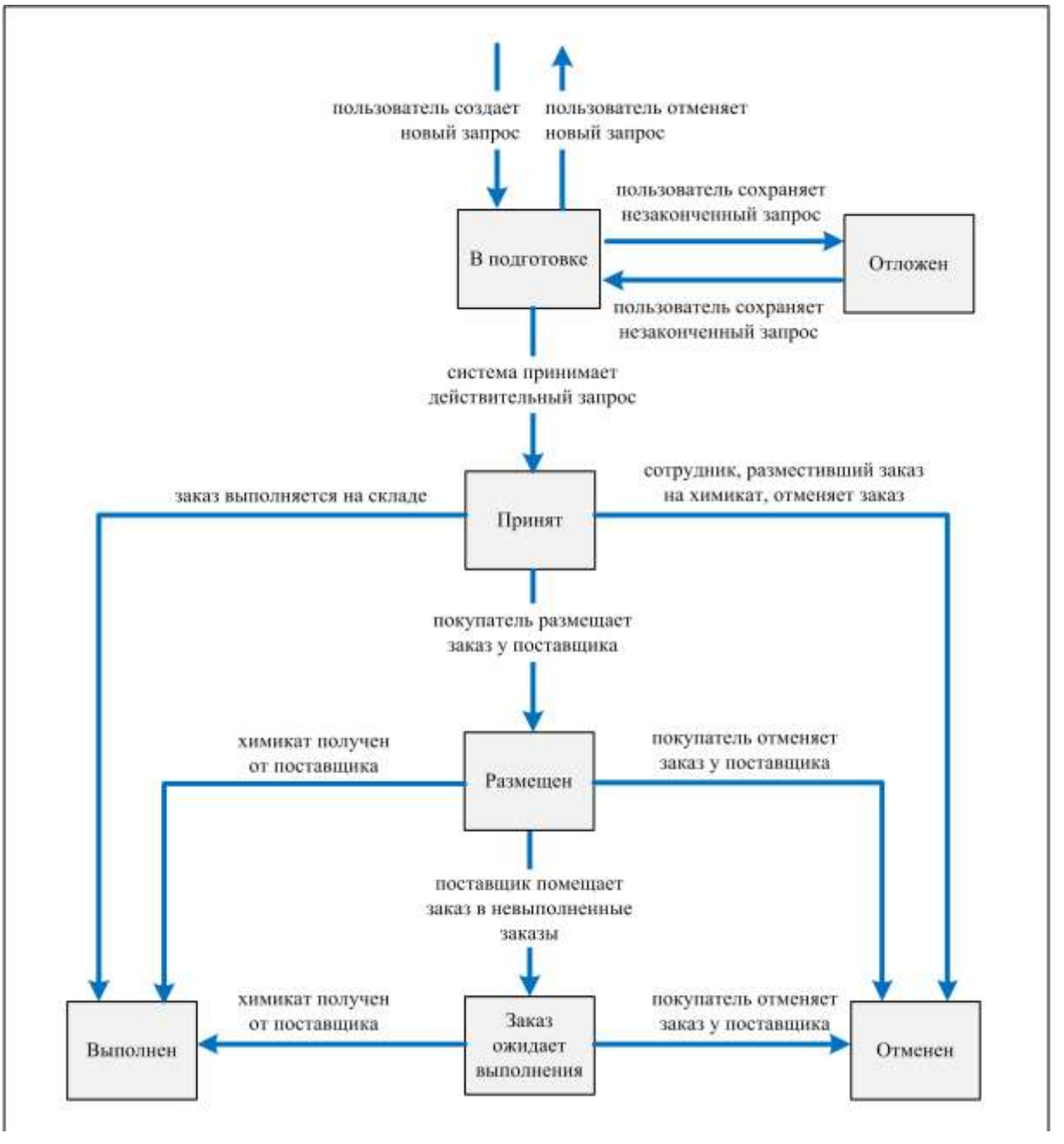


Рис. 11 -4. Диаграмма перехода состояний для запроса химиката для Chemical Tracking System

Карта диалогов

Пользовательский интерфейс также можно рассматривать как механизм с конечным числом состояний. Только один элемент диалога (такой, как меню, рабочая область, диалоговое окно, командная строка или дисплей сенсорного экрана) доступен в определенный момент времени для ввода информации пользователем. Пользователь может перейти к другим определенным элементам диалога, связанным с действием, которые он выполняет в активной области ввода. Количество возможных путей навигации в сложном графическом интерфейсе велико, однако оно конечно и, как правило, все возможности известны. Следовательно, множество пользовательских интерфейсов можно моделировать, применяя одну из форм диаграммы перехода состояния, которая называется ***карта диалогов*** (dialog map); (Wasserman, 1985; Wiegers, 1996a). Constantine и Lockwood (1999) описывают похожий прием — ***карту перемещений*** (navigation map), которую отличает более богатый набор условных обозначений для представления различных типов элементов взаимодействия и контекстных переходов.

Карта диалогов представляет дизайн пользовательского интерфейса на высоком уровне абстракции. На ней показаны элементы диалога в системе и навигация между ними, но не показан подробный вид экрана. Карта диалогов позволяет рассмотреть возможные концепции пользовательского интерфейса с учетом вашего понимания требований. Пользователям и разработчикам стоит изучить карту диалогов, чтобы выработать общее представление того, как пользователь может взаимодействовать с системой для выполнения задачи. Карты диалогов также полезны при моделировании визуальной архитектуры Web-сайта. Ссылки для перемещений, которые вы включаете Web-сайт, на картах диалогов изображаются в виде переходов. Карты диалогов связаны с архивными системными документами, в которые также включено краткое описание назначения каждого экрана (Leffingwell и Widrig, 2000).

Карты диалогов отражают сущность взаимодействий системы и пользователя и основные моменты решения задачи, без тормозящих работу команды деталей макета экрана. С помощью такой карты пользователи могут отследить отсутствующие, неправильные или ненужные переходы и, следовательно, отсутствующие, неправильные или ненужные требования. Абстрактная концептуальная карта диалогов, разработанная в ходе анализа требований, становится руководством для подробной разработки пользовательского интерфейса.

Также как и обычные диаграммы перехода состояния, карта диалогов показывает каждый элемент диалога как состояние (прямоугольник) и каждую допустимую возможность перемещения как переход (стрелка). Условие, инициирующее перемещение по пользовательскому интерфейсу, показано в виде текстового ярлыка на стрелке перехода. Существует несколько типов инициализирующих условий:

- действие пользователя, например нажатие функциональной клавиши или щелчок гиперссылки или кнопки диалогового окна;
- значение данных, такое, как недостоверная информация, в результате чего появляется сообщение об ошибке;
- системное условие, например отсутствие бумаги в принтере;
- некоторые комбинации вышеперечисленных, например ввод номера элемента меню и нажатие клавиши Enter,

Карты диалогов слегка напоминают диаграммы потоков, но у них другое назначение. Диаграммы потока ясно показывают этапы процесса и точки принятия решений, но не пользовательский интерфейс. В отличие от них, на карте диалогов *не* отображается процесс, выполняющийся по линиям перехода, которые соединяют один элемент диалога с другим. Выполнение решения (обычно это выбор пользователи) скрыто за экранами, которые показаны на карте диалогов в виде прямоугольников, а условия, в результате которых отображается тот или другой экран, описаны над стрелками переходов. Вы можете считать карту диалогов как противоположность — или дополнение — диаграммы потока.

Чтобы упростить карту диалогов, пропустите глобальные функции, такие, как нажатие клавиши F1 для вызова справки для каждого диалогового элемента. В разделе спецификации требований к ПО, посвященному пользовательскому интерфейсу, должно быть указано, что эта

функциональность будет доступна, но демонстрация множества экранов справки на карте диалогов вносит в модель беспорядок при незначительных преимуществах. Точно так же при моделировании Web-сайта не нужно включать стандартные для каждой страницы ссыпки перемещения. Вы также можете опустить транзакции, реализующие последовательность перемещений по Web-странице в обратном направлении, которые запускаются кнопкой Back Web-браузера.

Карта диалогов — это прекрасный способ представить взаимодействия действующего лица и системы, описываемые вариантом использования. Карта диалогов позволяет отобразить альтернативные направления в виде ответвлений от нормального направления. Я обнаружил, что наброски фрагментов карты диалогов на доске оказались особенно полезны на семинарах по сбору информации, касающейся вариантов использования, когда участники изучали последовательность действий пользователя и реакций системы при выполнении задачи.

В главе 8 описан вариант использования Chemical Tracking System под названием «Запрос химиката». Нормальное направление развития этого варианта использования подразумевает запрос контейнера с химикатом со склада. Альтернативное направление — запрос химиката у поставщика. Пользователь, размещающий запрос, хотел бы иметь возможность просматривать историю доступных контейнеров с этим химикатом, прежде чем выбрать один из них. На рис. 11-5 показана карта диалогов для этого несколько сложного варианта использования,

На первый взгляд эта диаграмма может показаться довольно запутанной, но в ней довольно легко разобраться, если попробовать за один раз отследить один прямоугольник и одну линию. Пользователь инициирует этот вариант использования, размещая заказ химиката из меню Chemical Tracking System. В карте диалогов это действие пользователя, обозначенное стрелкой в верхней левой части карты, направленной к прямоугольнику с названием «Список текущих запросов». Прямоугольник представляет основное рабочее пространство для этого варианта использования — список химикатов в текущем запросе пользователя. Стрелки, исходящие из этого прямоугольника на карте диалогов показывают все возможные перемещения — и, следовательно, функциональность — доступные пользователю:

- отменить весь запрос;
- передать к исполнению запрос, если в нем содержится хотя бы один химикат;
- добавить новый химикат к запросу;
- удалить химикат из списка.

В последней операции, удалении химиката, не участвует какой-либо еще элемент диалога, при этом просто обновляется список текущих запросов после того, как пользователь вносит изменения.

По мере перемещения по этой карте диалогов вы увидите элементы, отражающие остальные элементы варианта использования «Запрос химиката»:

- один путь для запроса химиката у поставщика;
- другой путь для выполнения запроса через склад химикатов;
- еще один возможный путь — просмотр истории контейнера, хранящегося на складе химикатов;

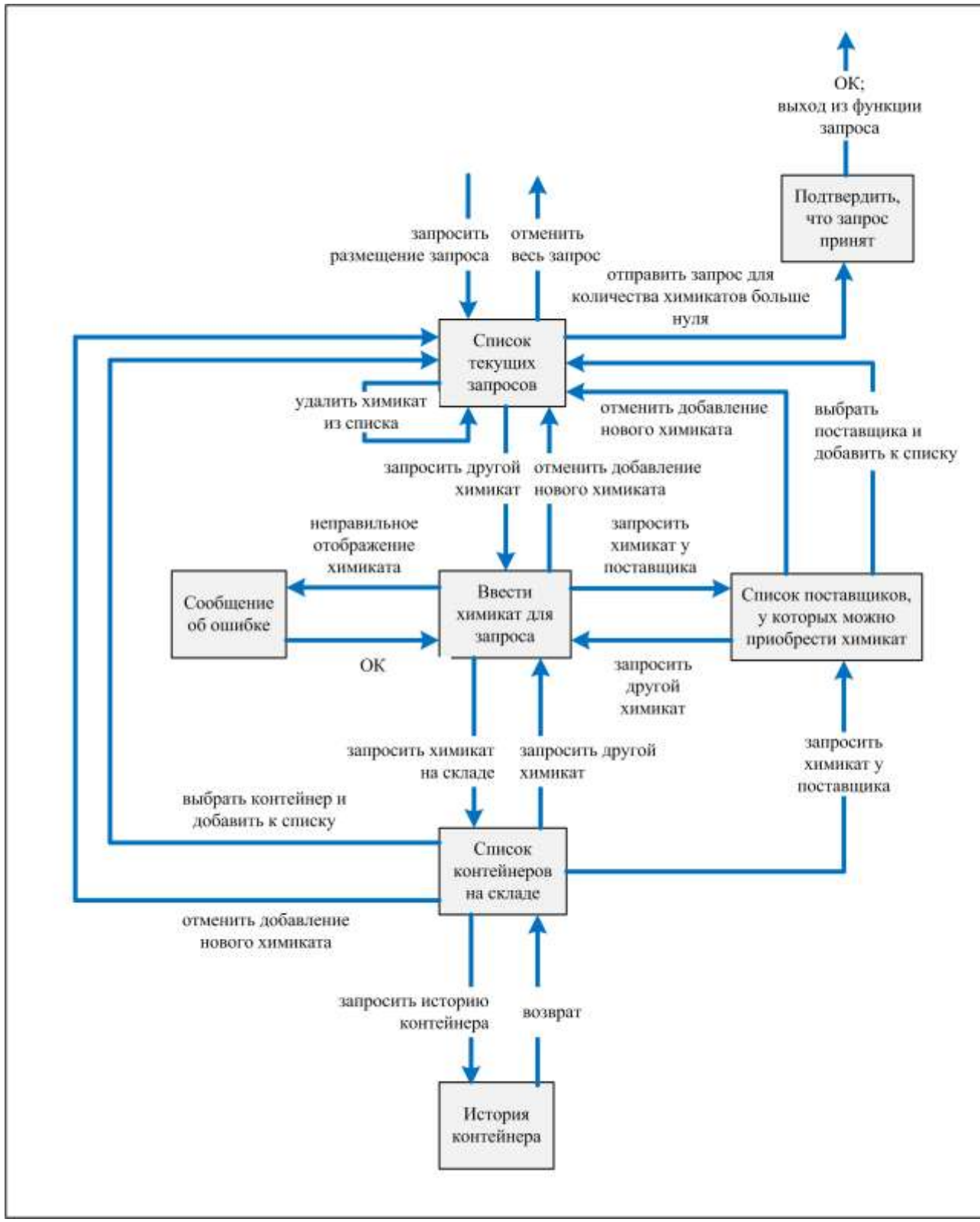


Рис. 11-5. Карта диалогов для варианта использования «Запрос химиката» в системе Chemical Tracking System

Некоторые переходы на карте диалогов позволяют пользователю откатить операцию. Пользователей раздражает, если они уже передумали, а им все-таки приходится завершать задачу. Карты диалогов позволяют облегчить и упростить работу, предлагая функции отката и отмены в стратегических точках.

Пользователь, изучающий карту диалогов, может обнаружить недостающее требование. Например, осторожный пользователь может захотеть подтвердить операцию, которая отменяет

весь запрос, чтобы избежать непроизвольной потери данных. Дешевле добавить такую функцию на этапе анализа, чем встраивать ее в уже законченный продукт, Поскольку карта диалогов представляет концептуальные элементы, участвующие во взаимодействии пользователя и системы, не пытайтесь зафиксировать все детали пользовательского интерфейса на этапе работы над требованиями. Лучшее применение этих моделей — помочь всем, кто заинтересован в проекте, выработать общее понимание предполагаемой функциональности системы.

Диаграмма классов

Объектно-ориентированная разработка ПО вытеснила структурный анализ и разработку, породив объектно-ориентированные анализ и разработку. Как правило, в предметной или рабочей области ***объекты*** (objects) соотносятся с объектами реального мира. Они представляют отдельные экземпляры, выведенные из родового шаблона, который называется ***класс*** (class). В описания классов входят атрибуты (данные) и действия, которые можно выполнять с этими атрибутами. ***Диаграмма классов*** (class diagram) — это графический способ отобразить классы, идентифицированные в ходе объектно-ориентированного анализа, и взаимоотношений между ними.

Для продуктов, разработанных с помощью объектно-ориентированных методов, не требуются уникальные приемы разработки требований. Дело в том, что требования отражают то, что пользователям необходимо сделать с помощью системы, и особенности предполагаемой функциональности, а не то, как она будет создана. Пользователям не придется вникать в объекты и классы. Однако если вы собираетесь разрабатывать систему с помощью объектно-ориентированных приемов, то анализ требований стоит начать с определения классов доменов, их атрибутов и поведения. Это упростит переход от анализа к проектированию, так как проектировщик сопоставляет объекты предметной области с системными объектами и детализирует атрибуты и операции каждого класса.

Стандартным языком объектно-ориентированного моделирования является унифицированный язык моделирования (Unified Modeling Language, UML) (Booch, Rumbaugh и Jacobson, 1999). На уровне абстракции, который годится для анализа требований, вы можете использовать систему обозначений UML в диаграмме классов, как показано на рис. 11-6 для части — вы правильно угадали — Chemical Trackinq System. Дизайнер переработает эти концептуальные диаграммы классов, не зависящие от особенностей реализации, в более подробные диаграммы классов для объектно-ориентированного проектирования и реализации. Взаимодействия классов и сообщения, которыми они обмениваются, демонстрируют диаграммы последовательностей и диаграммы сотрудничества, подробно о которых рассказано в Booch, Rumbaugh; Jacobson(1999) и Lauesen (2002).

На рис. 11-6 показано 4 класса — каждый в большом прямоугольнике: «Сотрудник, разместивший заказ на химикат», «Каталог поставщика», «Запрос химиката» и «Элемент строки запроса». Вы видите, что информация на этой диаграмме классов и данные на другой модели анализа, из тех, что показаны в этой главе, похожи (неудивительно — везде обсуждается одна и та же проблема). Сотрудник, разместивший заказ на химикат, показан на диаграмме «сущность-связь» на рис.11-2, где он представляет действующее лицо — член класса пользователей «Химики» или «Сотрудники склада химикатов». На диаграмме потоков данных на рис. 11-1 также видно, что оба эти класса могут размещать запросы химикатов. Кстати, не путайте ***класс пользователей*** и ***класс объектов***, невзирая на схожесть названий, они не связаны между собой.

Атрибуты (attributes), связанные с классом «Сотрудник, разместивший заказ на химикат», показаны в средней части прямоугольника, который обозначает их класс: имя, ID_сотрудника, Номер_отдела и Номер_офиса (правило применения заглавных букв довольно распространено в UML диаграммах). Это свойства или элементы данных, связанные с каждым объектом — членом класса «Сотрудник, разместивший заказ на химикат». Похожие атрибуты могут содержаться в определении хранилищ на диаграмме потоков данных или в словаре данных.

Операции (operations) — это службы, которые объект каждого класса может выполнять. Операции показаны в нижней части поля класса, как правило, за ними стоят пустые скобки. На диаграмме классов, представляющей проектирование, эти операции соответствуют функциям или

методам класса, а аргумент функции часто заключен в скобки. В модели анализа класса просто должно быть показано, что Сотрудник, разместивший заказ на химикат, может запрашивать химикаты, выполнять поиск по каталогам продавцов и получать контейнеры с химикатами. Операции на диаграмме классов весьма приблизительно соответствуют процессам, показанным в кружках на диаграммах потоков данных нижних уровней.

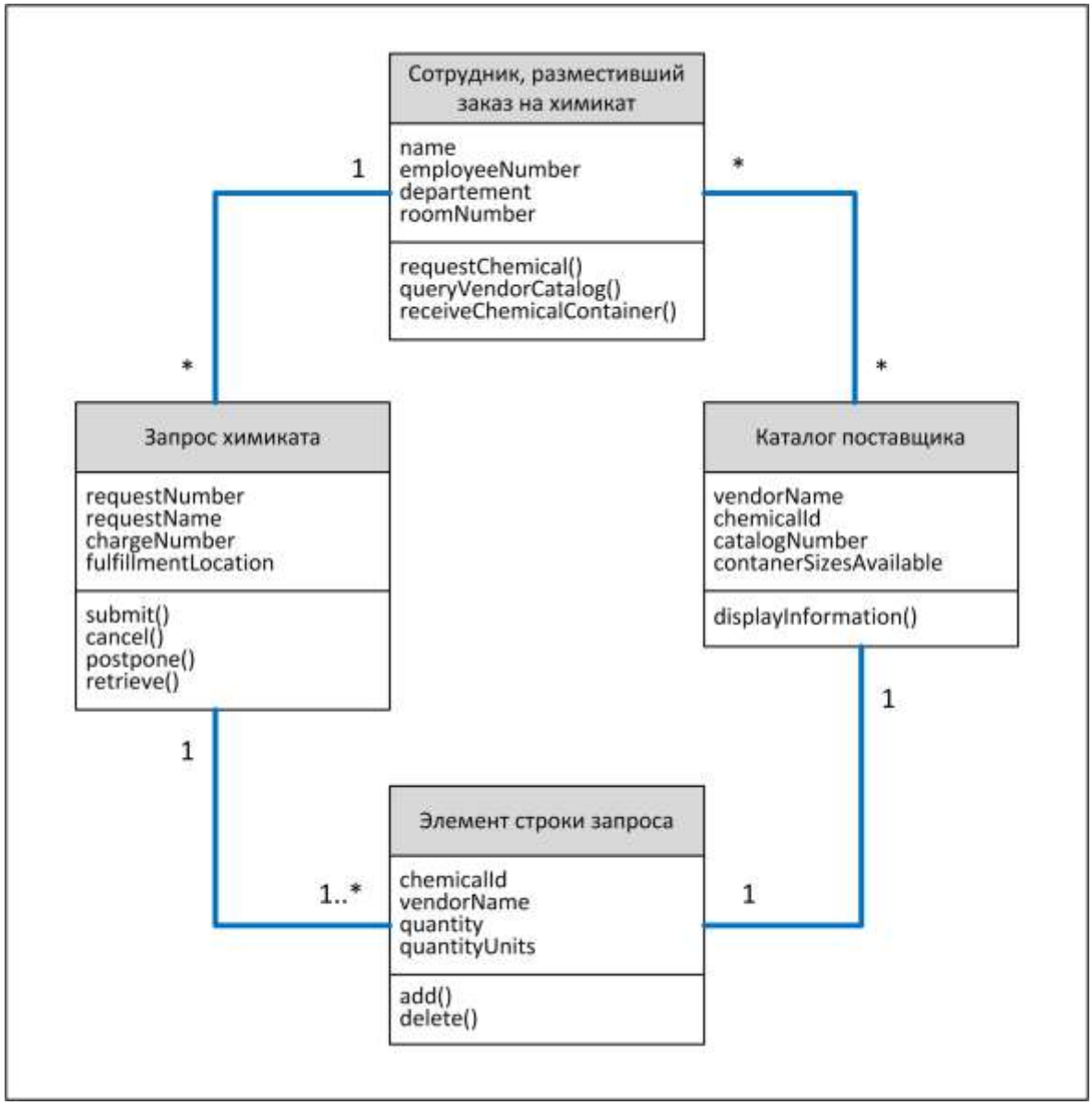


Рис. 11-6. Диаграмма классов для части Chemical Tracking, System

Линии, соединяющие прямоугольники классов на рис. 11-6, символизируют связи между классами. Цифры на них указывают на множественность связи, точно так же, как линии на диаграммах «сущность-связь» — на множественность взаимоотношений между объектами. На рис. 11-6 знак звездочки обозначает связь «один ко многим» между классами «Сотрудник, разместивший заказ на химикат» и «Запрос химиката»: один сотрудник может разместить множество запросов, но каждый запрос принадлежит только одному сотруднику.

Таблицы решений и деревья решений

Часто система ПО управляется сложной логикой, учитывающей различные комбинации условий, результатом которых является различное поведение системы. Например, если водитель нажимает кнопку ускорения на системе круиз-контроля автомобиля и автомобиль в данный момент двигается, то система увеличит скорость автомобиля, в противном случае команда игнорируется.

Для спецификации требований к ПО необходимы функциональные требования, в которых будет описано, что должна делать система при всех комбинациях условий. Однако условие легко пропустить, в результате чего пропускается и требование. Эти пробелы трудно заметить, просматривая текстовые спецификации вручную.

Таблицы решений и деревья решений — это два альтернативных приема для представления того, что система должна делать, когда в игру вступают сложные логика и решения (Davis, 1993). В ***таблице решений*** (decision table) перечислены различные значения для всех факторов, влияющих на поведение системы, и приведены ожидаемые действия системы в ответ на каждую комбинацию факторов. Факторы могут быть показаны либо как утверждения с различными условиями *true* и *false*, либо как вопросы с возможными ответами «да» или «нет». Естественно, вы также можете использовать таблицы решений с фак торами, которые имеют более двух возможных значений.

Ловушка

Не нужно создавать и таблицу решений, и дерево решений, чтобы показать один и тот же фрагмент информации; вполне достаточно одного из них.

В табл. 11-2 показана таблица решений с логикой, управляющей принятием или отклонением запроса нового химиката Chemical Tracking System. На решение влияет четыре фактора:

- авторизирован ли пользователь, создающий запрос;
- имеется ли химикат в наличии на складе или у поставщика;
- включен ли химикат в список опасных химикатов, для работы с которыми необходима специальная подготовка;
- есть ли у пользователя, создающего запрос, соответствующая подготовка для работы с этим типом опасного вещества.

Каждый из этих четырех факторов имеет два возможных условия, *true* или *false*. В принципе это увеличивает на 24 или 16 различные комбинации true/false для 16 вероятных отдельных функциональных требований. Однако на практике результатом многих комбинаций является одна и та же реакция системы. Если пользователь не авторизован для запроса химикатов, то система не примет запрос, и таким образом остальные условия несущественны (показаны прочерками в таблице решений). Таблица показывает, что результат различных логических комбинаций — лишь пять отдельных функциональных требований.

Таблица 11 -2. Пример таблицы решений для Chemical Tracking System

Номер требования					
Условие	**1**	**2**	**3**	**4**	**5**
Пользователь авторизован	false	true	true	true	true
Химикат есть в наличии	-	false	true	true	true
Химикат считается опасным	-	-	false	true	true
Сотрудник, разместивший заказ на химикат, прошел соответствующую подготовку	-	-	-	false	true
Действие					
Принять запрос			x		x
Отклонить запрос	·	x		x	

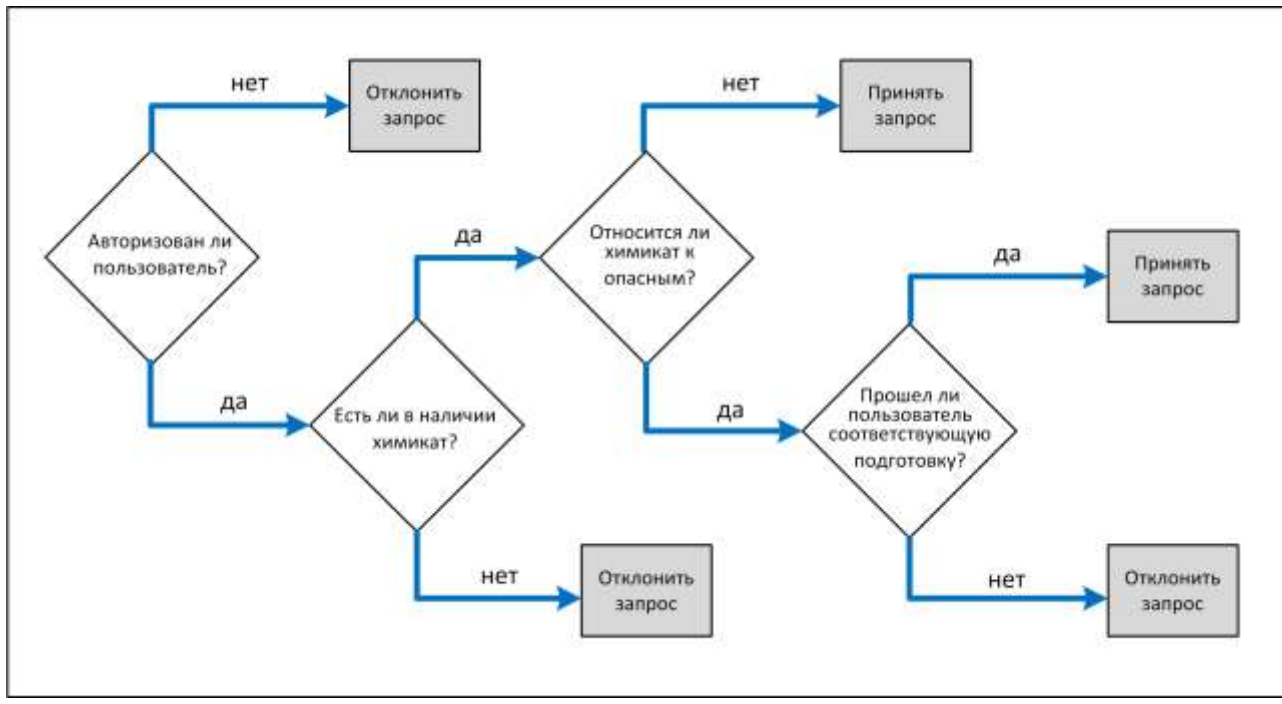


Рис. 11 -7. Пример дерева решений для Chemical Tracking System

На рис. 11-7 показано дерево решений, представляющее ту же логику. Пять прямоугольников обозначают пять возможных результатов принятия или отклонения запроса на химикат. Таблицы решений и деревья решений — это хорошие способы документации требований (или бизнес-правил), позволяющие не пропустить ни одну комбинацию условий. Даже сложная таблица или дерево решений более просты для чтения, чем повторяющиеся требования в текстовом виде.

Последнее напоминание

У каждого приема моделирования, описанного в этой книге, есть и свои преимущества, и свои ограничения. Эти приемы позволяют представить одни и те же области, поэтому вам не надо создавать для своего проекта все типы диаграмм. Например, если вы создаете диаграмму «сущность-связь» и словарь данных, не стоит рисовать диаграмму классов (или наоборот). Помните, что вы создаете модели анализа для того, чтобы обеспечить более высокий уровень понимания и взаимодействия, чем тот, что дает текстовая спецификация требований к ПО или любое другое представление требований. Старайтесь не стать догматиком и не участвовать в религиозных войнах, которые иногда случаются в мире методов и моделей разработки ПО. Вместе этого, используйте все доступные средства для того, чтобы как можно лучше объяснить требования к вашей системе.

Что теперь?

- Опробуйте на практике приемы моделирования, описанные в этой главе, задокументировав разработку существующей системы. Например, нарисуйте карту диалогов для банковского автомата или Web-сайта, которые вы используете;
- Определите часть спецификации требований к ПО, трудную для восприятия, или ту, где обнаружены дефекты. Выберите модель анализа, описанную в этой главе, которая подходит для представления этой части требований. Нарисуйте модель и оцените, помогла бы она решить проблему, создай вы ее раньше;
- В следующий раз, когда вам понадобится задокументировать определенные требования, выберите прием моделирования, который дополняет текстовое описание. Сделайте набросок модели на бумаге или доске один или два раза, чтобы убедиться, что вы на верном пути, затем воспользуйтесь коммерческими инструментами автоматизированного проектирования ПО, которые поддерживают условные обозначения модели, которую вы используете.

Глава 12 Обратная сторона функциональности: атрибуты качества ПО

«Привет, Фил, это снова Мария. У меня вопрос о новой служебной системе, которой вы сейчас занимаетесь. Как вы знаете, эта система работает на нашем мэйнфрейме и каждый отдел должен ежемесячно оплачивать используемое дисковое пространство и загрузку процессора. Похоже, что в новой системе файлы занимают в два раза больше места на диске, чем в старой. Что еще хуже, загрузка процессора за сессию в три раза выше, чем обычно. Вы можете объяснить мне, что происходит?»

«Конечно, Мария. Вспомните, вы хотели, чтобы в этой системе хранилось больше данных о каждом сотруднике, чем в предыдущей, поэтому, естественно, база данных стала значительно больше. Следовательно, ваша ежемесячная плата за использование дискового пространства выросла. Кроме того, вы и остальные сторонники продукта просили, чтобы с новой системой было удобнее работать, для чего мы сконструировали этот замечательный графический интерфейс. Однако при этом процессору требуется намного больше компьютерной мощности, чем в предыдущей системе с текстовым режимом отображения. Вот почему так возросла загрузка процессора. Но ведь с новой системой намного удобнее работать, не так ли?»

«Так-то оно так, но я и представить себе не могла, что ее работа будет обходиться так дорого. У меня из-за этого могут быть неприятности. Мой менеджер нервничает. При таких условиях он уже в апреле израсходует годовой бюджет, выделенный отделу на компьютерные нужды. Не могли бы вы исправить систему, чтобы снизить стоимость ее работы?» Фил расстроился. «Ничего исправлять не нужно. Новая система учета сотрудников — это именно то, что вы заказывали. Я предполагал, вы отдаете себе отчета том, что если вы храните больше данных или больше работаете на компьютере, то ваши затраты вырастут. Возможно, нам следовало обсудить это раньше, поскольку сейчас уже практически ничего не удастся сделать. Мне очень жаль».

Естественно, что пользователей, как правило, интересуют функциональные или поведенческие требования — то есть возможности, предоставляемые ПО, однако для успеха ПО недостаточно правильно реализовать соответствующую функциональность. Кроме того, пользователей волнует, насколько хорошо новая система будет работать. К характеристикам, описывающим это свойство системы, относятся легкость использования, быстрота запуска, количество сбоев и обработка неожиданных ситуаций. В целом они называются ***атрибутами качества*** *ПО* или ***факторами качества*** и считаются частью нефункциональных (или не поведенческих) требований к системе.

Довольно трудно дать определение атрибутам качества, однако часто именно они отличают продукт, которые просто работает так, как ожидалось, от продукта, который вызывает у клиентов восхищение. По мнению Роберта Шаррета (Robert Charette, 1990), «В реальных системах успех или неудачу проекта часто определяют именно нефункциональные требования, а не функциональные». В отличном ПО выдержан оптимальный баланс конкурирующих характеристик качества. Если в ходе сбора информации о требованиях вы досконально не выясните ожидания клиента, относящиеся к качеству, то вам крупно повезет, если продукт их удовлетворит. Но, как правило, более частый исход — разочарованные пользователи и расстроенные разработчики.

С технической точки зрения атрибуты качества влияют на важные решения, касающиеся архитектуры и проектирования; примером может служить распределение системных функций по различным компьютерам для достижения целей, связанных с производительностью или целостностью. Гораздо труднее и дороже перестраивать систему, чем спланировать необходимые параметры с самого начала.

Клиенты, как правило, высказывают свои ожидания о качестве продукта неявно, однако все же в ходе сбора информации удается выяснить основное. Хитрость заключается в том, чтобы

уловить, что же стоит за их рассуждениями о том, что система должна быть простой в обращении, быстрой, надежной или устойчивой к сбоям. Качество, во всех его проявлениях, должно быть определено и клиентами, и теми, кто создает, тестирует и поддерживает ПО. Вопросы, с помощью которых выясняются невысказанные ожидания клиентов, позволяют установить и качество продукта, и критерии проектирования, что поможет разработчикам создать отличный продукт.

Атрибуты качества

К атрибутам качества можно отнести несколько дюжин характеристик продукта (Charette, 1990), хотя для большинства проектов хватило бы всего нескольких. Если разработчикам известно, какие характеристики наиболее важны для успеха проекта, они могут выбрать соответствующие приемы, работая над архитектурой, проектированием и реализацией ПО, которые позволят достичь определенного качества (Glass, 1992; DeGrace и Stahl, 1993). Для классификации атрибутов качества применяются различные схемы (Boehm, Brown и Lipow, 1976; Cavano и McCall, 1978; IEEE 1992; DeGrace и Stahi, 1993). Один из способов классификации основан на разделении характеристик, которые проявляются в период выполнения, и тех, что не проявляются (Bass, Clements и Kazman, 1998). Согласно другому способу разделяются очевидные характеристики, главным образом важные для пользователей, от скрытых качеств, которые имеют значение для службы технической поддержки. Последние косвенно влияют на мнение клиента, так как упрощают возможные изменения продукта, его корректировку, проверку и переход на другие платформы.

В табл. 12-1 перечислено несколько атрибутов качества обеих категорий, которые необходимо принимать во внимание в любом проекте. Некоторые много значат для встроенных систем (эффективность и надежность), тогда как другие особенно важны для Интернет-приложений и приложений для мэйнфреймов (доступность, целостность и легкость в эксплуатации) или для настольных систем (способность к взаимодействию и удобство и простота использования). Для встроенных систем часто учитывают и другие важные атрибуты качества, например безопасность (о которой рассказывалось в главе 10), легкость и простота установки и обслуживания. Для Интернет-приложений также важен еще один атрибут — масштабируемость.

В идеальном мире каждая система имела бы максимально возможные значения всех этих атрибутов. Она была бы постоянно доступна и интуитивно понятна, не испытывала бы никаких сбоев, моментально предоставляла бы результаты, естественно, всегда корректные. Поскольку все перечисленное — утопия, советую вам выяснить с помо щью табл. 12-1, какие атрибуты наиболее важны для успеха вашего проекта. Затем разделите их на те, что важны пользователям, и те, что важны разработчикам, чтобы дизайнеры смогли принять соответствующие решения.

Таблица 12-1. Атрибуты качества ПО

Важны преимущественно для пользователей	**Важны преимущественно для разработчиков**
Доступность Эффективность Гибкость Целостность Способность к взаимодействию Надежность Устойчивость к сбоям Удобство и простота использования	Легкость в эксплуатации Легкость перемещения Возможность повторного использования Тестируемость

Для различных элементов ПО необходимы различные сочетания атрибутов качества. Для одних крайне важна эффективность, а для других — практичность. Выделите параметры качества, относящиеся к продукту в целом, и те, что важны только для определенных компонентов, классов пользователей или вариантов использования. Документируйте ваши глобальные пожелания, касающиеся качества продукта, в разделе 4.5 шаблона спецификации о требованиях к ПО, приведенного в главе 10, и ассоциируйте их с отдельными возможностями, вариантами

использования или функциональными требованиями.

Определение атрибутов качества

Большинство пользователей не знают ответов на такие вопросы, как «Каковы ваши требования к возможности взаимодействия?» или «Насколько надежным должно быть программное обеспечение?». При работе над Chemical Tracking System аналитики придумали несколько вспомогательных вопросов для каждого из атрибутов, которые они считали важными. Например, при исследовании целостности они заинтересовались, насколько важно не дать пользователям возможность просматривать заказы, которые были размещены другими пользователями, или должна ли у каждого пользователя быть возможность поиска по списку химикатов, находящихся на складе. Они опрашивали представителей пользователей, чтобы оценить каждый атрибут по шкале от 1 (не имеет никакого значения) до 5 (крайне важно). Ответы помогли аналитикам выделить наиболее важные атрибуты. Иногда у различных классов пользователей оказывались собственные предпочтения, тогда при разрешении любых конфликтов преимущество отдавалось привилегированному классу.

Затем аналитики поработали с пользователями, чтобы выработать определенные, измеряемые и поддающиеся проверке требования к каждому атрибуту (Robertson и Robertson, 1997). Если атрибуты качества нельзя проверить, то не удастся установить, достигнуты ли они. Следует указывать масштаб или единицы измерения для каждого атрибута и поставленной задачи, а также их минимальные и максимальные значения. Система обозначений Planguage, которая описывается далее в этой главе, поможет вам в этом. Если вы не знаете, как измерить все важные атрибуты качества, по крайней мере определите их приоритеты и предпочтения клиентов. Стандарт IEEE для Software Quality Metrics Methodology предоставляет прием для определения требований к качеству ПО в контексте общей системы измерения качества (IEEE, 1992).

Ловушка

При рассмотрении атрибутов качества не пренебрегайте мнением всех заинтересованных сторон, в том числе программистов по техническому обслуживанию.

Возможно, стоит спросить пользователей, как они представляют себе *неприемлемые* производительность, удобство и простоту использования, целостность и надежность. Это позволит определить свойства системы, которые противоречат ожиданиям пользователей о качестве, например возможность удаления файлов неавторизованными пользователями (Voas, 1999). Определяя неприемлемые характеристики — своего рода обратные требования — попробуйте разработать тесты, чтобы реализовать эти характеристики системы на практике. Если это вам не удастся, то, вероятно, вы достигли своих целей в отношении атрибутов. Такой способ особо важен для приложений, безопасность которых критически важна: негативные изменения их надежности или производительности аукнутся тяжелыми последствиями.

В заключительной части этого раздела кратко описан каждый атрибут качества из табл. 12-1 и приводятся некоторые примеры атрибутов (немного упрощенные) из различных проектов. Soren Lauesen (2002) предлагает множество отличных примеров требований, относящихся к качеству.

Атрибуты, важные для пользователей

Пользователи совершенно справедливо считают весьма важными атрибуты качества, описанные в этом разделе.

Доступность. Под доступностью понимается запланированное ***время доступности*** (uptime), в течение которого система действительно доступна для использования и полностью работоспособна. Формально доступность равна ***среднему времени наработки на отказ*** (mean time to failure, MTTF) системы, деленному на сумму среднего времени наработки на отказ и ожидаемого времени до восстановления системы после сбоя. На доступность также влияют периоды планового технического обслуживания. Некоторые авторы рассматривают доступность как совокупность

надежности, легкости в эксплуатации и целостности (Gilb, 1938).

Отдельные задачи более других зависят от времени, и пользователи будут страшно разочарованы (и даже разгневаны), если система не окажется доступной в нужный момент. Узнайте у клиентов, какой процент времени доступности им действительно необходим и есть ли периоды времени, когда доступность настоятельно необходима для бизнеса или выполнения задач, связанных с безопасностью. Для Web-сайтов и глобальных приложений, с которыми работают пользователи по всему миру, требования по доступности еще более сложны и важны. Одно из требований к доступности можно сформулировать так:

Доступность-1. *Система должна быть доступна как минимум на 99,5% по рабочим дням, с 6:00 до полуночи по местному времени и доступна как минимум на 99,95% по рабочим дням, с 16:00до 18:00 по местному времени.*

Как и другие приведенные здесь примеры, это требование несколько упрощено. Оно не определяет уровень производительности ***вовремя доступности***. Считается ли система доступной, если в режиме с плохими характеристиками в ней может работать только один пользователь сети? Записывайте требования к качеству, чтобы их можно было измерить и добиться четкого соглашения между аналитиком требований, командой разработчиков и клиентами.

Цена качества

Не указывайте 100% для таких атрибутов качества, как надежность или доступность, поскольку такие результаты недостижимы, а стремление достичь их обойдется дорого. Одна компания указала в требованиях к системе по обслуживанию цеха доступность, равную 100%, 24 часа в день, 365 дней в году. Пытаясь добиться указанного уровня доступности, было решено установить две компьютерные системы, чтобы обновление ПО выполнялось на машине, не работающей в данный момент. Это дорогостоящее решение оказалось дешевле, чем прекращение производства крайне прибыльного товара.

Эффективность. Эффективностью называется показатель того, насколько эффективно система использует производительность процессора, место на диске, память или полосу пропускания соединения (Davis, 1993). Эффективность связана с производительностью еще одним классом нефункциональных требований, который обсуждается далее в этой главе. Если система тратит слишком много доступных ресурсов, пользователи заметят снижение производительности — видимого показателя неэффективности. Недостаточная производительность раздражает пользователей, которые ожидают вывода на экран результата запроса к базе данных. Но проблемы производительности кроме того, ставят под удар безопасность, например, при перегрузке системы контроля процессов реального времени. Определите минимальную конфигурацию оборудования, при которой удается достичь заданных эффективности, пропускной способности и производительности. Чтобы позволить нижний предел в случае непредвиденных условий и определить последующий рост, вы можете воспользоваться такой формулировкой:

Эффективность-1. *Как минимум 25% пропускной способности процессора и оперативной памяти, доступной приложению, не должно использоваться в условиях запланированной пиковой нагрузки.*

Типичные пользователи не формулируют требования к эффективности в таких технических терминах. Максимум, на что они способны, так это упомянуть время отклика или заполнение пространства на диске. Дело аналитика — задать вопросы, которые выявят ожидания пользователей о приемлемом снижении производительности, возможных пиках нагрузки и ожидаемом росте.

Поспешишь — людей насмешишь

Одна крупная корпорация разрабатывала сложную графическую модель магазина для их компонентов электронного бизнеса. Покупатель мог посетить электронный магазин на Web-сайте, просмотреть предлагаемые услуги и приобрести различные товары. Графика была великолепной, модель — качественной, но быстродействие оказалось ужасным. Пользовательский интерфейс отлично работал у разработчиков, использовавших высокоскоростное Интернет-соединение с локальными серверами. К сожалению, при стандартных подключениях через модем на скорости 14,4 или 28,8 кбит/с, которые в то время использовало большинство пользователей, огромные файлы изображений загружались невероятно медленно. Все без исключения бета-тестеры теряли интерес к сайту еще до того, как главная страница успевала полностью загрузиться. Увлекшись графической моделью, разработчики не подумали об ограничениях операционной среды, эффективности и требованиях к производительности. Пришлось отказаться от этого решения уже после его завершения — дорогостоящий урок, подтвердивший важность обсуждения атрибутов качества ПО в начале проекта

Гибкость. Этот атрибут также называют *расширяемостью, дополняемостью, наращиваемостью* или *растяжимостью*. Гибкость показывает с какой легкостью в продукт удается добавить новые возможности. Если ожидается, что при разработке придется вносить множество улучшений, стоит выбрать такие решения, которые позволят увеличить гибкость ПО. Этот атрибут важен для продуктов, в качестве модели разработки которых выбрано улучшение и повтор успешных выпусков или развитие прототипа. Для проекта, в котором н принимал участие, были сформулированы следующие цели, касающиеся реализации гибкости в ПО:

Гибкость-1. *Программист по техническому обслуживанию, не менее шести месяцев работающий с продуктом, должен уметь подключать новое устройство для создания печатных копий, что предусматривает изменение кода и тестирование, не более чем за час рабочего времени.*

Проект нельзя считать неудачным, если программисту требуется 7 минут, чтобы установить новый принтер, следовательно, это требование допускает некоторую степень свободы. Если бы мы не указали это требование, возможно, разработчики выбрали бы такой вариант проектирования, при котором установка нового устройства в системе заняла бы слишком много времени. Записывайте требования к качеству в формате, который предусматривает возможность их измерения.

Целостность. Целостность, которая включает в себя безопасность, о чем уже говорилось в главе 10, связана с блокировкой неавторизированного доступа к системным функциям, предотвращением потери информации, антивирусной защитой ПО и защитой конфиденциальности и безопасности данных, введенных в систему. Целостность очень важна для интернет-приложений. Пользователи систем электронной коммерции хотят обезопасить данные своих кредитных карточек. Посетители Web-сайтов не желают, чтобы приватная информация о них или список посещаемых ими сайтов использовались не по назначению, а поставщики услуг доступа к Интернету хотят защититься от атак типа «отказ в обслуживании» и прочих хакерских атак. В требованиях к целостности нет места ошибкам. Используйте следующие точные термины для формулирования требований к целостности: проверка идентификации пользователя, уровни привилегий пользователя, ограничения доступа или определенные данные, которые должны быть защищены, Вот как можно' сформулировать требования к целостности:

Целостность-1. *Только пользователи, обладающие привилегиями уровня Аудитор, должны иметь возможность просматривать транзакции клиентов.*

Как и многие другие требования к целостности, данное помимо прочего считается

ограничивающим бизнес-правилом. Неплохо бы знать логическое обоснование ваших требований к атрибутам качества и исследовать их происхождение, например политику управления. Избегайте указывать требования к целостности в виде ограничений проектирования, как, скажем, требования к паролю для контроля доступа. Реальное требование должно ограничивать доступ к системе неавторизированных пользователей; пароли - всего лишь один из способов (хотя и самый распространенный) выполнения этой задачи, Основанное на выбранном подходе к идентификации пользователей, это базовое требование к целостности повлияет на определенные функциональные требования, которые реализуют функции аутентификации в системе.

Способность к взаимодействию. Способность к взаимодействию показывает, каким образом система обменивается данными или сервисами с другими системами. Чтобы оценить способность к взаимодействию, вам необходимо знать, какие приложения клиенты будут применять совместно с вашим продуктом и обмен каких данных предполагается. Пользователи Chemical Tracking System привыкли рисовать химические структуры, с помощью нескольких коммерческих инструментов, поэтому они выдвинули следующее требование к способности взаимодействия:

Способность к взаимодействию-1. *Chemical Tracking System должна иметь возможность импортировать любые допустимые химические структуры из пакетов ChemiDraw (версии 2.3 или более ранней) и Chem-Struct (версии 5 или более ранней).*

Вы также могли бы указать данное требование как требование к внешнему интерфейсу и определить стандартные форматы файлов, которые способна импортировать Chemical Tracking System. В качестве альтернативы можно определить несколько функциональных требований, относящихся к операции импорта. Однако иногда, рассматривая систему с точки зрения атрибутов качества, вы выясняете, что некоторые определенные требования не заданы. Клиенты не указали данную потребность при обсуждении внешнего интерфейса или функциональности системы. Как только аналитик задал вопрос о других системах, с которыми придется взаимодействовать Chemical Tracking System, сторонник продукта немедленно упомянул два пакета для рисования химических структур.

Надежность. Надежностью называется вероятность работы ПО без сбоев в течение определенного периода времени (Musa, lannino и Okumoto, 1987). Иногда одной из характеристик надежности считают устойчивость к сбоям. Для измерения надежности ПО используют такие показатели, как процент успешно завершенных операций и средний период времени работы системы до сбоя. Определите количественные требования к надежности, основываясь на том, насколько серьезными окажутся последствия сбоя и оправдана ли цена повышения надежности. Системы, для которых требуется высокая надежность, следует проектировать с высокой степенью возможности тестирования, чтобы облегчить выявление недостатков, отрицательно влияющих на надежность.

Однажды моя команда разрабатывала ПО для управления лабораторным оборудованием, предназначенным для опытов с редкими дорогостоящими химикатами, длящихся целый день. Пользователям требовался программный компонент, который обеспечил бы надежность проведения экспериментов. Другие системные функции, такие как периодическая запись в журнал данных о температуре, были не столь важными. Требования к надежности для данной системы звучали так:

Надежность-1. *Не более пяти из тысячи начатых экспериментов могут быть потеряны из-за сбоев ПО.*

Устойчивость к сбоям. Однажды клиент компании, разрабатывающей устройства для измерений, пожелал, чтобы их следующий продукт «был, как танк». Сотрудникам компании так понравилось сравнение, что они ввели в обиход новый, слегка ироничный атрибут качества — «как танк». Его стали применять, когда речь шла об устойчивости к сбоям, которую еще иногда называют ***отказоустойчивостью*** (fault tolerance). Под устойчивостью к сбоям понимают уровень, до которого система продолжает корректно выполнять свои функции, несмотря на неверный ввод данных, недостатки подключенных программных компонентов или компонентов оборудования или неожиданные условия работы. Устойчивое к сбоям ПО легко восстанавливается после различных проблем и «не замечает» ошибок пользователей. Выясняя требования к устойчивости работы ПО, спросите пользователей, какие ошибочные ситуации возможны при работе с системой и как

система должна на них реагировать. Вот один из примеров требования к устойчивости к сбоям:

Устойчивость к сбоям-1. *Если при работе с редактором произошел сбой и пользователь не успел сохранить файл, то редактор должен восстановить все изменения, внесенные раньше, чем за минуту до сбоя, при следующем запуске программы данным пользователем.*

Несколько лет назад я занимался разработкой программного компонента многократного использования Graphics Engine, который преобразовывал файлы с графическими схемами и выводил изображение на назначенное устройство вывода (Wiegers. 1996b). Несколько приложений, которым было необходимо создавать графики, обращались к Graphics Engine. Поскольку разработчики не могли контролировать, какие именно данные приложения передают в Graphics Engine, важным атрибутом качества была названа устойчивость к сбоям системы. Одно из наших требований звучало так:

Устойчивость к сбоям-2. *Для всех параметров, описывающих графики, должны быть указаны значения по умолчанию, которые ядро Graphics Engine будет использовать в случае, если данные входного параметра указаны неверно или отсутствуют.*

Выполнение этого требования позволит избежать сбоя программы, если, например, приложение запрашивает цвет, который плоттеру не удается воспроизвести. Graphics Engine использует значение по умолчанию — черный цвет — и продолжит работу. Тем не менее это можно рассматривать как сбой ПО, поскольку конечный пользователь не получил желаемый цвет. Однако такой подход снизил серьезность последствий сбоя — вместо краха программы получен неправильный цвет, что является примером отказоустойчивости.

Удобство и простота использования. Также называется ***легкостью использования и инженерной психологией***. Этот атрибут связан с массой факторов, которые составляют основу того, что пользователи часто описывают ***как дружелюбие к пользователю***. Но в лексиконе аналитиков и разработчиков нет термина «дружественное ПО», они говорят о ПО, которое спроектировано для эффективного и необременительного использования. В последнее время вышло несколько книг, посвященных разработке удобных в использовании программных систем, например Constantine и Lockwood (1999) и Nielsen (2000). Удобство и простота использования измеряется усилиями, требуемыми для подготовки ввода данных, эксплуатации и вывода конечной информации.

Аналитики требований для Chemical Tracking System задавали представителям пользователей такие вопросы, как «Насколько важна для вас возможность быстрого и простого запроса химикатов?» и «Сколько времени вы готовы отвести на запрос?» Все это — несложные исходные точки для определения многих характеристик, которые могут сделать ПО удобным в работе. При обсуждении этого атрибута удается выявить параметры, поддающиеся измерению, например:

Удобство и простота использования-1. *Пользователь, прошедший соответствующую подготовку, должен иметь возможность выбрать требуемый химикат из каталога поставщика в среднем за четыре и максимум за шесть минут.*

Узнайте, должна ли новая система соответствовать каким-либо стандартам или соглашениям, касающимся пользовательского интерфейса, и должен ли последний быть совместим с другими часто используемыми системами. Вы можете сформулировать такое требование следующим образом:

Удобство и простота использования-2. *Всем функциям меню File должны соответствовать быстрые клавиши, нажимаемые одновременно с Ctrl. Командам меню, которые также присутствуют в меню File пакета Microsoft Word XP, должны соответствовать те же быстрые клавиши, что и в Word.*

Кроме того, удобство и простота использования определяется и тем, насколько легко новые или непостоянные пользователи научатся работать с продуктом. Простота обучения также поддается исчислению и измерению:

Удобство и простота использования-3. *Химик, который прежде никогда не использовал Chemical Tracking System, должен не более чем 30 минут разобраться, как правильно запросить химикат.*

Атрибуты, важные для разработчиков

В этом разделе описываются атрибуты качества, которые в первую очередь важны для разработчиков ПО и специалистов по техническому обслуживанию.

Легкость в эксплуатации. Этот атрибут показывает, насколько удобно исправлять ошибки или модифицировать ПО. Легкость в эксплуатации зависит от того, насколько просто разобраться в работе ПО, изменять его и тестировать, и тесно связано с гибкостью и тестируемостью. Этот показатель крайне важен для продуктов, которые подвергаются частым изменениям, и тех, что создаются быстро (и, возможно, с экономией на качестве). Легкость в эксплуатации измеряют, используя такие термины, как среднее время, требуемое для разрешения проблемы, и процент корректных исправлений. Для Chemical Tracking System одно из требований к легкости в эксплуатации сформулировано таким образом:

Легкость в эксплуатации-1 *Программист, занимающийся техническим обслуживанием ПО, должен модифицировать существующие отчеты, чтобы привести их в соответствие с изменениями в положениях федерального правительства в области химии, затратив на разработку не более 20 рабочих часов.*

При работе над проектом Graphics Engine мы понимали, что нам придется часто вносить исправления в ПО, чтобы оно соответствовало потребностям пользователей. Мы указали примерно такие критерии, чтобы разработчикам удалось создать ПО, более легкое в эксплуатации:

Легкость в эксплуатации-2. *Вложенность вызываемых функций не должна превышать два уровня.*

Легкость в эксплуатации-3. *Для каждого программного модуля непустые комментарии в соотношении к исходному коду должны составлять как минимум 0,5.*

Ставьте такие задачи разработчикам осторожно, иначе те, лишившись самообладания, начнут действовать формально, выполняя букву, но не суть задачи. Поработайте с программистами, занимающимися техническим обслуживанием, чтобы понять, какие качества исходного кода облегчат им внесение изменений или исправление недостатков.

Для аппаратных устройств со встроенным ПО часто имеются требования к легкости в эксплуатации. Одни из них относятся к выбору проектирования ПО, в то время как другие влияют на проектирование оборудования. Например последнее можно сформулировать так:

Легкость в эксплуатацин-4. *Проектирование принтера позволяет сертифицированному ремонтнику заменить кабельный шнур печатающей головки не более чем за 10 минут, датчик ленты не более чем за 5 минут и привод ленты не более чем за 5 минут.*

Легкость перемещения. Мерой ее измерения можно считать усилия, необходимые для перемещения ПО из одной операционной среды в другую. Некоторые практики считают возможность интернационализации и локализации продукта высшей степенью его мобильности. Приемы разработки ПО, которые делают легким его перемещение очень схожи с теми, что применяют, чтобы сделать ПО многократного используемым (Glass, 1992). Обычно мобильность продукта или не имеет никакого значения, или крайне важна для успеха проекта. Важно определить те части продукта, которые необходимо легко перемещать в другие среды, и описать эти целевые среды. Затем разработчики выберут способы разработки и кодирования, которые увеличат мобильность продукта.

Например, одни компиляторы определяют размер типа данных integer в 16 бит, а другие — 32 бит. Чтобы выполнить требования к мобильности, программисту надо в символической форме определить тип данных *WORD* как 16-битное целое без знака и использовать тип данных *WORD* вместо целочисленного типа данных, принятого в компиляторе по умолчанию. Таким образом гарантируется, что все компиляторы будут одинаково обращаться к элементам данных типа *WORD,* что сделает работу системы предсказуемой в различных операционных средах.

Возможность повторного использования. Постоянная задача разработки ПО — возможность повторного использования — показывает усилия, необходимые для преобразования программных компонентов с целью их дальнейшего применения в других приложениях. Затраты на разработку ПО с возможностью повторного использования значительно выше, чем на создание компонента, который будет работать только в одном приложении. Оно должно быть модульным, хорошо задокументированным, не зависеть от конкретных приложения и операционной среды, а

также обладать некоторыми универсальными возможностями. Цели многократного использования сложно количественно измерить. Укажите, какие элементы новой системы необходимо спроектировать таким образом, чтобы упростить их повторное применение, или укажите библиотеки компонентов многократного использования, которые необходимо создать дополнительно к проекту. Например:

Возможность повторного использования-1. Функции ввода химических структур должны быть спроектированы таким образом, чтобы их удавалось повторно использовать на уровне объектного кода в других приложениях, построенных с учетом международных стандартов для представления химических структур.

Тестируемость. Этот атрибут также называют ***проверяемостью,*** он показывает легкость, с которой программные компоненты или интегрированный продукт можно проверить на предмет дефектов. Такой атрибут крайне важен для продукта, в котором используются сложные алгоритмы и логика или имеются тонкие функциональные взаимосвязи. Тестируемость также важна в том случае, если продукт необходимо часто модифицировать, поскольку предполагается подвергать его частому регрессивному тестированию, чтобы выяснить, не ухудшают ли внесенные изменения существующую функциональность.

Поскольку я и команда разработчиков твердо знали, что нам придется тестировать Graphics Engine много раз в период его неоднократных усовершенствований, мы включили в спецификацию следующую директиву, касающуюся тестируемости ПО:

Тестируемость-1. *Максимальная цикломатическая сложность модуля не должна превышать 20.*

Цикломатическая сложность (cyclomatic complexity) — это количество логических ответвлений в модуле исходного кода (McCabe, 1982).

Чем больше ответвлений и циклов в модуле, тем тяжелее его тестировать, понимать и поддерживать. Конечно, проект не будет провален, если значение цикломатической сложности одного из модулей достигнет 24, однако наша директива указала разработчикам уровень качества, к которому следует стремиться. Если бы ее (она представлена как требование к качеству) не было, не факт, что разработчики при написании своих программ приняли бы во внимание такую характеристику, как цикломатическая сложность. В результате код программы мог оказаться так запутан, что его тщательное тестирование и дополнение оказалось бы практически невозможным, а отладка вообще стала бы кошмаром.

Требования к производительности

Требования к производительности определяют, насколько быстро и качественно система должна выполнять определенные функции. Они определяют такие параметры, как скорость (например, время отклика БД), пропускная способность (количество транзакций в секунду), мощность (нагрузка при совместном использовании) и распределение по времени (интенсивные запросы реального времени). Жесткие требования к производительности сильно влияют на стратегию разработки ПО и выбор оборудования, поэтому определите задачи, касающиеся производительности, для соответствующей операционной среды. Все пользователи хотят, чтобы их приложение запускалось моментально, но реальные требования к производительности различаются для функции проверки орфографии в программе подготовки текстов и для радиолокационной системы наведения ракеты. Требования к производительности также должны учитывать снижение производительности при такой перегрузке, как девятый вал звонков в службу спасения 911. Вот несколько простых требований к производительности:

Производительность-1. *Цикл контроля температуры должен быть полностью выполнен за 80 миллисекунд.*

Производительность-2. *Интерпретатор должен проводить в минуту разбор как минимум 5000 операторов, не содержащих ошибок.*

Производительность-3. *Каждая Web-страница должна загружаться не более чем за 15 секунд при модемном соединении со скоростью 50 кбит/с.*

Производительность-4. *Авторизация запроса на получение денег из банкомата не должна длиться более 10 секунд.*

Ловушка

Не забудьте обдумать, как вы будете оценивать продукт на соответствие атрибутам качества. Непроверяемые требования к качеству ничем не лучше непроверяемых функциональных требований.

Определение нефункциональных требований с помощью языка Planguage

Некоторые из атрибутов качества, перечисленных в этой главе, являются неполными или неспециализированными — очень трудно выразить их одной-двумя короткими фразами. Вам не удастся оценить, отвечает ли продукт неточно сформулированным требованиям к качеству или нет. Кроме того, упрощенные задачи, связанные с качеством или производительностью, могут оказаться невыполнимыми. Максимальное время отклика, равное двум секундам для запроса к БД, замечательно работает при несложном поиске в локальной БД, но абсолютно невыполнимо при соединении шести связанных таблиц, размещенных на серверах, которые расположены в разных местах.

Для решения проблемы неявных и неполных нефункциональных требований консультант Tom Gilb (1988; 1997) разработал *Planguage,* язык планирования с большим набором ключевых слов, который позволяет точно устанавливать атрибуты качества и другие задачи проекта (Simmons, 2001). Далее показано, как выразить требование к производительности с помощью всего лишь нескольких из множества ключевых слов Planguage. Этот пример является версией требований на языке Planguage к производительности из главы 10: «95% запросов БД каталога должны быть выполнены в течение 3 секунд на однопользовательском компьютере Intel Pentium 4, 1,1 ГГц под управлением Microsoft Windows XP, когда доступны как минимум 60% системных ресурсов».

TAG. Производительность. Время отклика на запрос.

AMBITION. Быстрый отклик на запросы БД на пользовательской платформе.

SCALE. Время (в секундах) между нажатием клавиши Enter и щелчком ОК для отправки запроса и началом отображения результатов запроса.

METER. Тестирование с использованием секундомера, выполненное на 250 тестовых запросах, представляющих определенный операционный профиль использования.

MUST. Не более 10 секунд на 98% запросов <— Специалист выездной службы поддержки.

PLAN. Не более 3 секунд для запросов 1-ой категории, 8 секунд для всех остальных запросов.

WISH. Не более двух секунд для всех запросов.

Основная пользовательская платформа DEFINED. Процессор Intel Pentium 4 1,1 ГГц, оперативная память 128 Мб, под управлением ОС Microsoft Windows XP с установленным пакетом QueryGen версии 3.3, однопользовательский компьютер, свободно как минимум 60% системных ресурсов, другие приложения не выполняются.

В каждом требовании присутствует уникальный ***тэг*** (tag), или метка. ***Цель*** (ambition) устанавливает цель или задачу для системы, которая порождает данное требование. ***Масштаб*** (scale) определяет единицы измерения, а ***счетчик*** (meter) описывает точно, как выполнить измерения. Все заинтересованные стороны должны одинаково хорошо понимать, каким образом измеряется производительность. Предположим, пользователь проще воспринимает систему измерений, которая основана на времени от нажатия клавиши «Enter» до вывода всех результатов запроса, чем до начала отображения результатов, как указано в примере, Разработчик может заявить, что требование удовлетворено, а пользователь будет утверждать обратное. Точно выраженные требования к качеству и системе измерений помогут предотвратить такого рода недоразумения.

Вы можете указать несколько желаемых количественных величин, которые необходимо измерять. Критерий ***must*** (обязательство) определяет наименьший достижимый уровень.

Требование не удовлетворено до тех пор, пока не будут удовлетворены все условия ***must***, таким образом, это условие должно быть обосновано в бизнес-терминах. Можно указать требование ***must*** другим способом. Для этого нужно определить условие ***fait*** (неудача) (еще одно ключевое слов языка Planguage), например: «Более 10 секунд ожидания для более 2% всех запросов». Величина ***plan*** (план) указывает номинальное значение, а ***wish*** (пожелание) представляет собой идеальный результат. Кроме того, покажите источник требуемой производительности, например, упомянутое выше условие ***must*** (обязательство) указал специалист выездной службы поддержки. Для облегчения чтения любые специализированные термины, используемые в операторах языке Planguage, заданы как ***defined*** (определяемые).

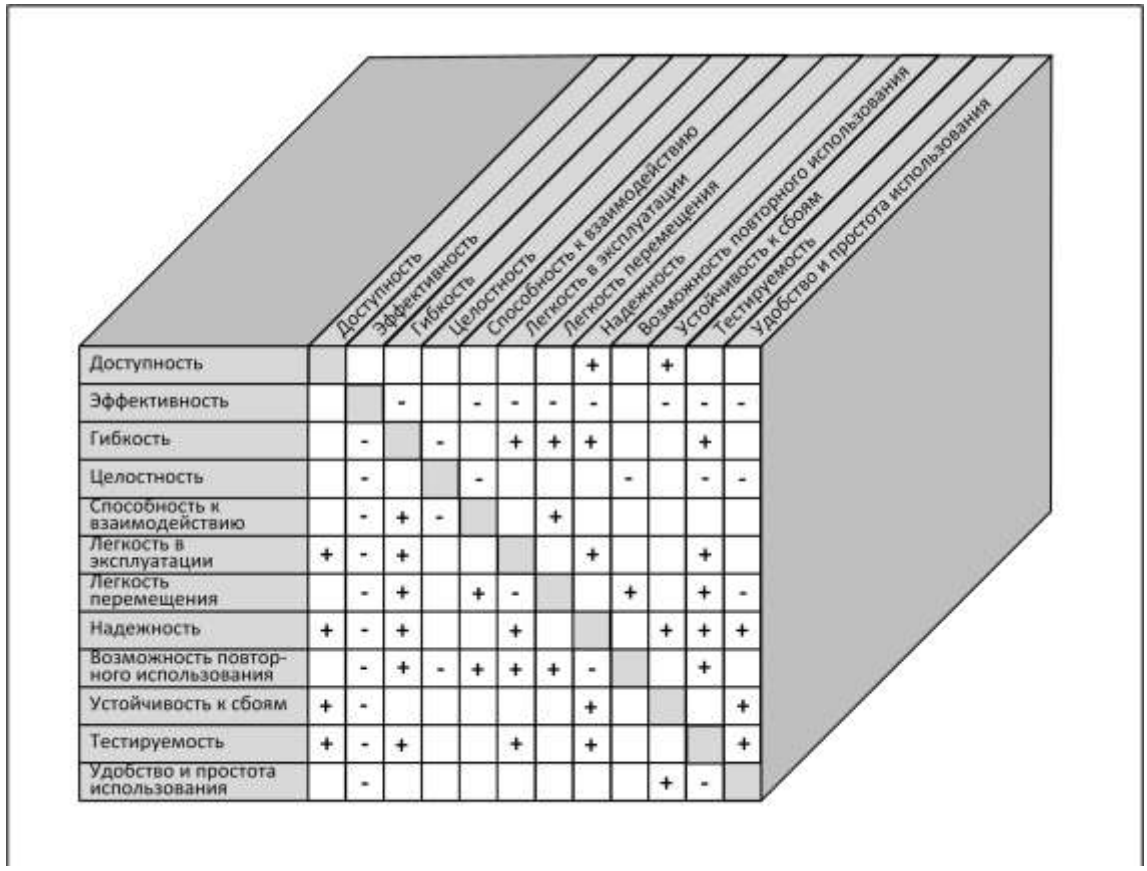
Planguage определяет множество дополнительных ключевых слов, с помощью которых удается добиться более гибкой и точной спецификации требований. Так как язык Planguage, его словарь и синтаксис еще развиваются, рекомендую обратиться за свежей информацией на Web-сайт *http://www.gilb.com*. Planguage представляет собой мощное средство для точной формулировки атрибутов качества и требований к производительности. Указав несколько уровней для ожидаемого результата, вы получите более широкое представление о требованиях к качеству, чем с помощью простых конструкций, таких, как «черное-белое» и «да-нет»,

Компромиссы для атрибутов

Для определенных комбинаций атрибутов компромиссы неизбежны. Пользователи и разработчики должны решить, какие атрибуты важнее других и при принятии решений соблюдать эти приоритеты. На рис. 12-1 показаны типичные взаимосвязи атрибутов качества, перечисленных в табл. 12-1, однако вам следует знать, что возможны исключения (Charette, 1990; IEEE, 1992; Glass, 1992) Знак в ячейке означает, что увеличение величины атрибута в соответствующей строке позитивно влияет на атрибут в соответствующем столбце. Например, при повышении легкости перемещения программного компонента повышается гибкость ПО, упрощается подключение к компонентам другого ПО и возможность повторного использования и тестирования.

Знак «-» в ячейке означает, что увеличение величины атрибута в этой строке негативно влияет на атрибут в соответствующем столбце. Пустая ячейка означает, что атрибут в этой строке оказывает незначительное влияние на атрибут в столбце. Эффективность оказывает негативное влияние на большинство атрибутов. Если вы напишите компактную и быструю программу, используя хитрости кодирования и учитывая некоторые побочные эффекты при выполнении, вероятнее всего эту программу будет сложно поддерживать и исправлять, а также переносить на другие платформы. Аналогично, системы, которые были оптимизированы для удобства использования или спроектированы как гибкие, пригодные для многократного применения и способные взаимодействовать с другими программными компонентами или компонентами оборудования, часто грешат проблемами производительности. При создании чертежей средствами универсального компонента Graphics Engine, о котором говорилось ранее в этой главе, производительность понизилась по сравнению с приложением, где использовался нестандартный код для обработки графики. Вам необходимо найти баланс между возможным ухудшением производительности и ожидаемой выгодой от предлагаемого решения, чтобы убедиться, что компромисс, на который вы идете, разумен.

Матрица на рис. 12-1 не симметрична, потому что влияние при увеличении атрибута А на атрибут В не обязательно такое же, как влияние, которое оказывает увеличение атрибута В на атрибут А. Так, из рис. 12-1 видно, что модификация системы с целью повышения эффективности не оказывает эффекта на целостность. Но повышение целостности вероятнее всего весьма отрицательно отразится на эффективности, поскольку системе придется выполнять множество проверок аутентификации пользователей, зашифрованных данных, проверок на наличие вирусов и контрольных точек данных.



	Доступность	Эффективность	Гибкость	Целостность	Способность к взаимодействию	Легкость в эксплуатации	Легкость перемещения	Надежность	Возможность повторного использования	Устойчивость к сбоям	Тестируемость	Удобство и простота использования
Доступность								+		+		
Эффективность			-		-	-	-	-		-	-	-
Гибкость		-		-		+	+	+			+	
Целостность		-			-				-		-	-
Способность к взаимодействию		-	+	-			+					
Легкость в эксплуатации	+	-	+					+			+	
Легкость перемещения		-	+		+	-			+		+	-
Надежность	+	-	+			+				+	+	+
Возможность повторного использования		-	+	-	+	+	+	-			+	
Устойчивость к сбоям	+	-						+				+
Тестируемость	+	-	+			+		+				+
Удобство и простота использования		-								+	-	

Рис. 12-1. Позитивные и негативные взаимосвязи некоторых атрибутов качества

Чтобы оптимизировать баланс атрибутов продукта, в процессе сбора информации вам следует определить, задокументировать важнейшие атрибуты качества и расставить для них приоритеты. При определении атрибутов качества для проекта, используйте рис. 12-1 для того, чтобы не поставить перед собой несовместимые цели. Как это, например, может быть, показано далее.

- Не пытайтесь максимизировать удобство и простоту использования, если ПО должно работать на нескольких платформах (мобильность)
- Нелегко полностью проверить требования к целостности систем с: высоким уровнем безопасности. Для универсальных компонентов многократного использования или тех, что предполагаются для связи с другими приложениями, можно в некоторой степени снизить требования безопасности.
- Крайне надежный код окажется менее эффективным из-за постоянного выполнения им проверок данных и поиска ошибок.

Как правило, слишком ограничивая возможности системы или определяя конфликтующие требования, вы тем самым усложняете задачу разработчикам — им гораздо труднее удовлетворить все требования заказчика.

Реализация нефункциональных требований

Проектировщики и программисты должны определить наилучший способ удовлетворения требования для каждого атрибута качества и производительности. Хотя атрибуты качества относятся к нефункциональным требованиям, они помогают сформулировать функциональные требования, определить направления разработки или техническую информацию, которая позволяют реализовать продукт желаемого качества. В табл. 12-2 перечислены некоторые категории технической информации, которые могут быть выведены с помощью атрибутов качества. Например, в медицинское устройство со строгими требованиями к доступности может входить источник резервного питания (архитектура) и функциональные требования для

визуального или звукового оповещения, когда продукт работает на резервном питании. Это преобразование требований к качеству, имеющих значение для пользователей или для разработчиков, в соответствующую техническую информацию является частью процесса разработки требований и проектирования высокого уровня.

Таблица 12-2. Преобразование требований к качеству в техническую документацию

Типы атрибутов качества	Категория технической информации
Целостность, способность к взаимодействию, устойчивость к сбоям, легкость и простота использования, безопасность	Функциональное требование
Доступность, эффективность, гибкость, производительность, надежность	Архитектура системы
Способность к взаимодействию, легкость и простота использования	Ограничения проектирования
Гибкость, легкость в эксплуатации, легкость перемещения, надежность, возможность повторного использования, тестируемость, легкость и простота использования	Руководство по проектированию
Легкость перемещения	Ограничение реализации

Что теперь?

- Определите несколько атрибутов качества для пользователей из табл. 12-1, которые могут пригодиться для вашего текущего проекта. Сформулируйте несколько вопросов о каждом атрибуте, которые помогут пользователям сформулировать их ожидания. Основываясь на ответах пользователей, письменно сформулируйте одну или несколько задач для каждого важного атрибута.
- Перепишите несколько примеров атрибутов качества из этой главы с помощью Planguage, при необходимости для наглядности делая некоторые допущения. Удается ли вам выразить эти требования к качеству более точно и недвусмысленно с помощью Planguage?
- Напишите одно из ваших собственных требований к атрибуту качества с помощью Planguage. Попросите клиента, разработчика и представителя отдела тестирования оценить, является ли Planguage-версия более информативной или нет? Изучите ожидания пользователей, касающиеся качества системы, на предмет возможных противоречий и устраните их. Привилегированные классы пользователей должны иметь наибольшее влияние при принятии компромиссов.
- Отследите, как атрибуты качества влияют на функциональные требования, ограничения дизайна и реализации или выбор архитектуры и дизайна

Глава 13 Прототипы как средство уменьшения риска

«Шэрон, сегодня я хотел бы поговорить с вами о требованиях покупателей из Отдела закупок к новой Chemical Tracking System, — начал Лори, аналитик требований. — Вы можете рассказать мне, что система должна делать?» «Ничего себе, я даже не знаю, как к этому подступиться, — ответила Шэрон озадаченно. — Я не знаю, как описать мои потребности, но узнаю, когда увижу».

«Узнаю, когда увижу», — фраза, от которой в жилах аналитиков требований стынет кровь. Сразу в воображении возникает картина: команда разработчиков воплощает свои лучшие идеи в разработке ПО, увидев которое пользователи говорят: "Нет, не то, попробуйте еще раз». Действительно, представить будущую программную систему и ясно изложить требования к ней — нелегкая задача. Многим людям сложно описать свои потребности, не имея перед собой чего-то осязаемого, без образца, да и критиковать гораздо легче, чем создавать.

Создание прототипов ПО делает требования более реальными, приближает варианты использования к «жизни» и закрывает пробелы в вашем понимании требований. Прототипы предоставляют пользователям экспериментальную модель или первоначальный срез новой системы, стимулируя их мышление и катализируя обсуждение требований. Обсуждение прототипов на ранних стадиях процесса разработки помогает заинтересованным в проекте лицам прийти к общему пониманию требований к системе, что уменьшает риск недовольства заказчиков.

Даже при применении методов разработки требований из предыдущих глав отдельные их части могут быть неопределенными или неясными, как для клиентов, так и для разработчиков, или и тех, и других.

Если вы не исправите эти проблемы, разница в ожиданиях между видением продукта пользователями и пониманием разработчиков гарантирована. Трудно точно представить поведение нового ПО при чтении текстовой документации или изучении аналитических моделей. Пользователи более готовы экспериментировать с прототипом (что весело), чем читать спецификацию требований к программному обеспечению (что скучно). Услышав от пользователей «узнаю, когда увижу», подумайте, как вы можете помочь им наглядно представить свои нужды (Boehm, 2000). Однако если ни один из участников не представляет себе, что же должны создать разработчики, проект обречен.

Прототип (prototype) имеет множество значений, поэтому ожидания участников процесса прототипирования могут весьма различаться. Прототип самолета действительно летает — ведь это первый вариант настоящего самолета. Напротив, прототип ПО — это только часть или образец реальной системы — он может в принципе не делать ничего полезного. Прототипы ПО могут быть действующими моделями или статичными образцами; детализированным изображением экрана или его эскизами; наглядной демонстрацией или примерами реальной функциональности; имитацией или подражанием (Constantine и Lock-wood, 1999; Stevens и др., 1998). В этой главе описаны различные виды прототипов, их использование во время разработки требований к ПО и способы эффективного внедрения макетирования в процесс разработке ПО (Wood и Kang, 1992).

Что такое прототип и для чего он нужен

Прототип ПО — это частичная или возможная реализация предлагаемого нового продукта. Прототипы позволяют решать три основные задачи:

- **прояснение и завершение процесса формулировки требований.** Используемый в качестве инструмента формулировки требований прототип представляет собой предварительную версию части системы, понимание которой вызывает затруднения. Оценка прототипа пользователями указывает на ошибки в формулировке требований, которые можно исправить без больших затрат до создания реального продукта;
- **исследование альтернативных решений.** Прототип, как инструмент конструирования, позволяет заинтересованным в проекте лицам исследовать различные варианты реализации взаимодействия пользователей, оптимизировать

удобство работы и оценить возможные технические приемы. Прототипы позволяют на рабочих образцах показать, насколько осуществимы требования;

- **создание конечного продукта.** Используемый в качестве инструмента разработки прототип — не что иное, как функциональная реализация первичных элементов системы, которую можно превратить в готовый продукт, осуществляя последовательную цепочку небольших циклов разработки.

Основная цель создания прототипа — устранение неясностей на ранних стадиях процесса разработки. Именно исходя из них, следует решать, для каких частей системы необходим прототип и что вы надеетесь выяснить, оценивая его. Прототип полезен для выявления и устранения двусмысленных и неполных утверждений в требованиях. Пользователи, менеджеры и другие заинтересованные в проекте лица, не обладающие техническими знаниями, считают, что прототип дает им понимание некой конкретики, пока реальный продукт документируется и разрабатывается. Прототипы, особенно наглядные, легче понять, чем технический жаргон разработчиков.

Горизонтальные прототипы

Когда люди говорят «прототип ПО», они обычно имеют в виду ***горизонтальный прототип*** (horizontal prototype) предполагаемого интерфейса пользователя. Его также именуют ***поведенческим прототипом*** (behavioral prototype) или моделью. Горизонтальным прототип называется потому, что в нем не реализуются все слои архитектуры и нюансы системы, но воплощаются некоторые особенности интерфейса пользователя. Он позволяет пользователям исследовать поведение предполагаемой системы в тех или иных ситуациях для уточнения требований, а также выяснить, смогут ли они с помощью системы, основанной на прототипе, выполнять свою работу.

Горизонтальный прототип, подобно кинофильму, создает видимость функциональности, не обеспечивая ее действительного воплощения. Он показывает внешний вид экранов пользовательского интерфейса и позволяет осуществлять частичную навигацию между ними, но не содержит никакой или почти никакой действительной функциональности. Это как типичный вестерн: ковбой входит в салун, а затем выходит из конюшни, но он не заказывает выпивку и не видит лошадей, потому что за фасадами декораций ничего нет.

Горизонтальные прототипы демонстрируют функциональные возможности, которые будут доступны пользователю, внешний вид пользовательского интерфейса (цвета, планировку, графику, элементы управления) и структуру доступа к информации (структуру навигации). Перемещение между объектами интерфейса возможно, но вместо некоторых элементов пользователь увидит лишь краткое сообщение о том, что будет здесь находиться. Информация, появляющаяся в ответ на запрос к базе данных, может быть случайной или постоянной, а содержание отчетов — жестко закодированным. Старайтесь использовать реальные данные в образцах отчетов, диаграмм и таблиц— это увеличит достоверность прототипа как модели реальной системы.

Горизонтальный прототип не выполняет полезной работы, хотя иногда и кажется, что должен. Зачастую имитации вполне достаточно, чтобы пользователи могли решить, нет ли каких-либо упущений, неверных или ненужных функций. Некоторые прототипы выражают концепцию реализации конкретных вариантов использования, как ее видят разработчики. Оценивая прототип, пользователи могут указать на альтернативные возможности их реализации, пропущенные шаги взаимодействия или дополнительные условия исключений.

Работая с горизонтальным прототипом, пользователь должен концентрироваться на общих требованиях и последовательности выполняемых действий, не отвлекаясь на особенности внешнего вида элементов экрана (Constantine, 1998). На этой стадии не беспокойтесь о точном расположении элементов экрана, шрифтах, цветах, рисунках и элементах управления. Время детализации интерфейса пользователя наступит после того, как прояснятся требования к системе и будет определена общая структура интерфейса.

Вертикальные прототипы

Вертикальный прототип (vertical prototype), также называемый ***структурным*** прототипом (structural prototype) или проверкой концепции, воплощает срез функциональности приложения от

интерфейса пользователя до сервисных функций. Вертикальный прототип действует как настоящая система, поскольку затрагивает все уровни ее реализации. Разрабатывайте вертикальный прототип всякий раз, когда сомневаетесь в осуществимости и стабильности предполагаемого подхода к архитектуре системы или когда хотите оптимизировать алгоритмы, оценить предлагаемую схему базы данных или проверить критически важные временные требования. Чтобы получить значимые результаты, вертикальные прототипы создают, используя средства разработки в среде, аналогичной среде разработки. Вертикальные прототипы используются для исследования критически важных требований к интерфейсу и времени исполнения, а также для сокращения рисков на стадии проектирования системы.

Однажды я работал с командой, которая занималась реализацией необычной клиент-серверной архитектуры для стратегии перехода от мэйнфрейм-системы к программной среде, основанной на серверах и рабочих станциях, работающих под UNIX и объединенных в сеть, (Thompson и Wiegers, 1995). Вертикальный прототип, реализовавший небольшую часть клиента пользовательского интерфейса (на мэйнфрейме) и соответствующую часть функциональности сервера (на рабочей станции под UNIX), позволил нам оценить коммуникационные компоненты, производительность и надежность предлагаемой архитектуры. Эксперимент прошел успешно, как и построение решения на основе этой архитектуры.

Одноразовые прототипы

Прежде чем создавать прототип, примите четкое и ясное решение, прекратите ли вы работать с ним после оценки или сделаете его частью выпускаемого продукта. Создайте ***одноразовый прототип*** (throwaway prototype) или ***исследовательский прототип*** (exploratory prototype), чтобы ответить на вопросы, разрешить неясности и улучшить требования к ПО (Davis, 1993). Если вы решили, что после выполнения задачи прекратите работу с прототипом[2], стройте его как можно более быстро и дешево. Чем больше усилий вы вложите в прототип, тем труднее будет участникам проекта отказаться от него.

Создавая одноразовый прототип, разработчики пренебрегают большинством известных им методов конструирования качественного ПО. В одноразовом прототипе предпочтение отдается быстроте реализации и модификации, а не устойчивости, надежности, производительности и долговременному удобству сопровождения. Поэтому внимательно следите, чтобы низкокачественный код одноразового прототипа не попал в окончательный продукт. В противном случае пользователи и персонал технического обслуживания будут страдать от последствий этого в течение всего срока эксплуатации продукта.

Одноразовый прототип наиболее уместен, когда команда разработчиков сталкивается с неизвестностью, двусмысленностью, неполнотой или неопределенностью в требованиях к ПО. Разрешение этих вопросов уменьшает риск продолжения разработки. Прототип, помогающий пользователям и разработчикам визуально представить реализацию требований, позволяет выявить пробелы в документации, а также решить, годятся ли требования для создания продукта, поддерживающего необходимые бизнес-процессы.

> **Ловушка**
>
> Не делайте прототип более сложным, чем это необходимо для целей его создания. Не поддавайтесь искушению - или давлению пользователей - добавить в прототип больше функций.

На рис. 13-1 показана последовательность шагов разработки от вариантов использования до детализированного проекта интерфейса пользователя через построение одноразового прототипа. Каждое описание варианта использования включает последовательность действий субъекта и

[2] Не стоит уничтожать прототип, если возможно его повторное применение в будущем. Тем не менее он не должен стать частью окончательного продукта. Поэтом назовите его нереализуемый прототип.

реакцию системы, которые вы можете смоделировать посредством схемы диалогов, чтобы отразить особенности архитектуры интерфейса пользователя. Одноразовый прототип представляет элементы диалога в виде экранов, меню и диалоговых окон. Когда его оценивают пользователи, возможны изменения в описании вариантов использования (если, например, обнаруживается альтернативный путь) или изменения в карте диалогов. Каждый раз, когда требования уточняются и создаются наброски экранов, элементы интерфейса пользователя оптимизируется для удобства и простоты использования. Такой метод постепенного уточнения обходится дешевле, чем если вы скакнете прямо от описаний вариантов использования к законченному интерфейсу пользователя, а потом обнаружите серьезные проблемы в требованиях, исправление которых потребует глобальной переделки продукта.

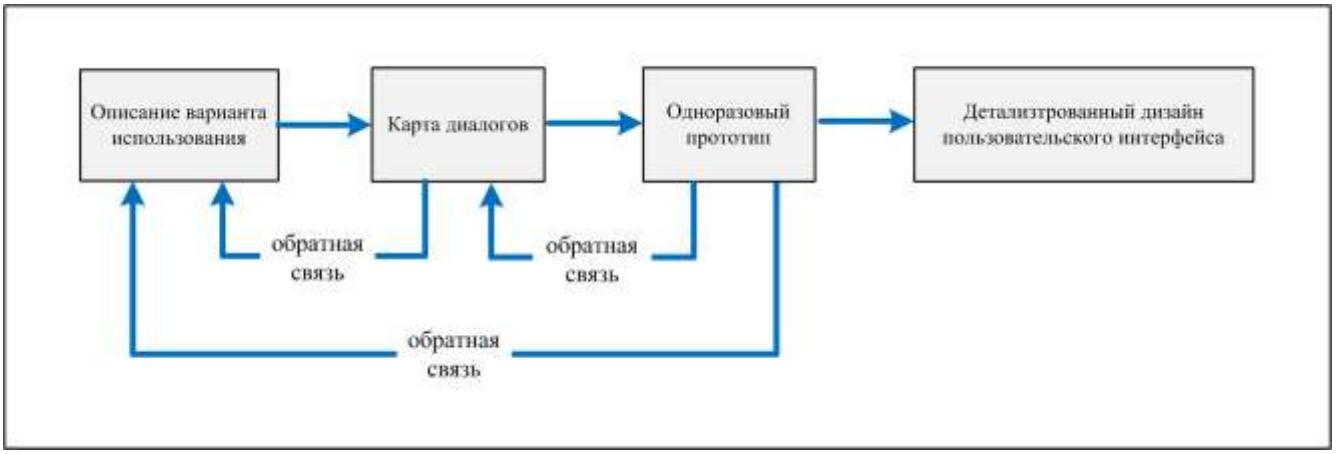


Рис. 13-1. Последовательность действий от вариантов использования к дизайну интерфейса пользователя через одноразовый прототип

Эволюционные прототипы

В отличие от одноразового прототипа, ***эволюционный прототип*** (evolutionary prototype) представляет собой прочный архитектурный «фундамент» для постепенного создания окончательного продукта — по мере прояснения требований. ***Эволюционное протатипирование*** — один из компонентов модели спирального цикла разработки ПО (Boehm, 1988) и некоторых процессов разработки объектно-ориентированного ПО (Kruchten. 1996). В отличие от одноразовых прототипов, создаваемы «быстро и начерно», для построения эволюционного прототипа необходимо с самого начала использовать отличный и качественный код. Поэтому на конструирование эволюционного прототипа требуется больше времени, чем на одноразовый прототип, моделирующий те же свойства системы. Эволюционный прототип следует создавать в расчете на легкий рост и частое расширение, поэтому разработчики должны уделять большое внимание архитектуре и принципам проектирования. При создании такого прототипа ни в коем случае не стоит экономить на качестве.

Считайте первый цикл в создании эволюционного прототипа пробным выпуском, реализующим хорошо понятную и неизменную часть требований. Уроки, извлеченные из реакции пользователей на тестирование и первоначальное использование продукта, учитываются при модификации в следующем цикле. Окончательный продукт — это кульминация серии эволюционных прототипов. Такие прототипы позволяют пользователям быстро получить действующие образцы. Эволюционные прототипы хорошо подходят для приложений, которые, как 1вы знаете, со временем будут расширяться; к ним относятся, например, проекты постепенной интеграции различных информационных систем.

Эволюционное прототипирование годится и для проектов разработки Интернет-приложений. В одном из таких проектов, которым я руководил, моя команда создала серию из четырех прототипов на основе требований, полученных в результате анализа вариантов использования. Каждый прототип оценивали несколько пользователей, и мы вносили изменения на основе их

ответов на наши вопросы. Исправления, которые мы внесли после оценки четвертого прототипа, стали' окончательным вариантом нашего Интернет-сайта.

На рис. 13-2 показано несколько способов комбинирования различных видов прототипирования. Например, информацию, полученную в результате выполнения серии одноразовых прототипов, вы можете использовать для уточнения требований, которые затем шаг за шагом реализовать через последовательность эволюционных прототипов. Другой способ, показанный на рис. 13-2, — применение горизонтального одноразового прототипа для прояснения требований перед разработкой окончательной версии дизайна пользовательского интерфейса, в то время как одновременно с помощью вертикального прототипирования проверяется архитектура системы, компоненты приложения и алгоритмы ядра. Что вам совершенно точно не удастся, так это превратить намеренно низкокачественный одноразовый прототип в удобный для сопровождения выверенный код, необходимый для построения окончательной системы. Кроме того, рабочие прототипы, которые демонстрируют возможности продукта нескольким пользователям, привлекаемым к разработке системы, скорее всего нельзя масштабировать для тысяч пользователей без серьезных архитектурных изменений. В табл. 13-1 суммированы некоторые стандартные варианты применения одноразовых, эволюционных, горизонтальных и вертикальных прототипов.

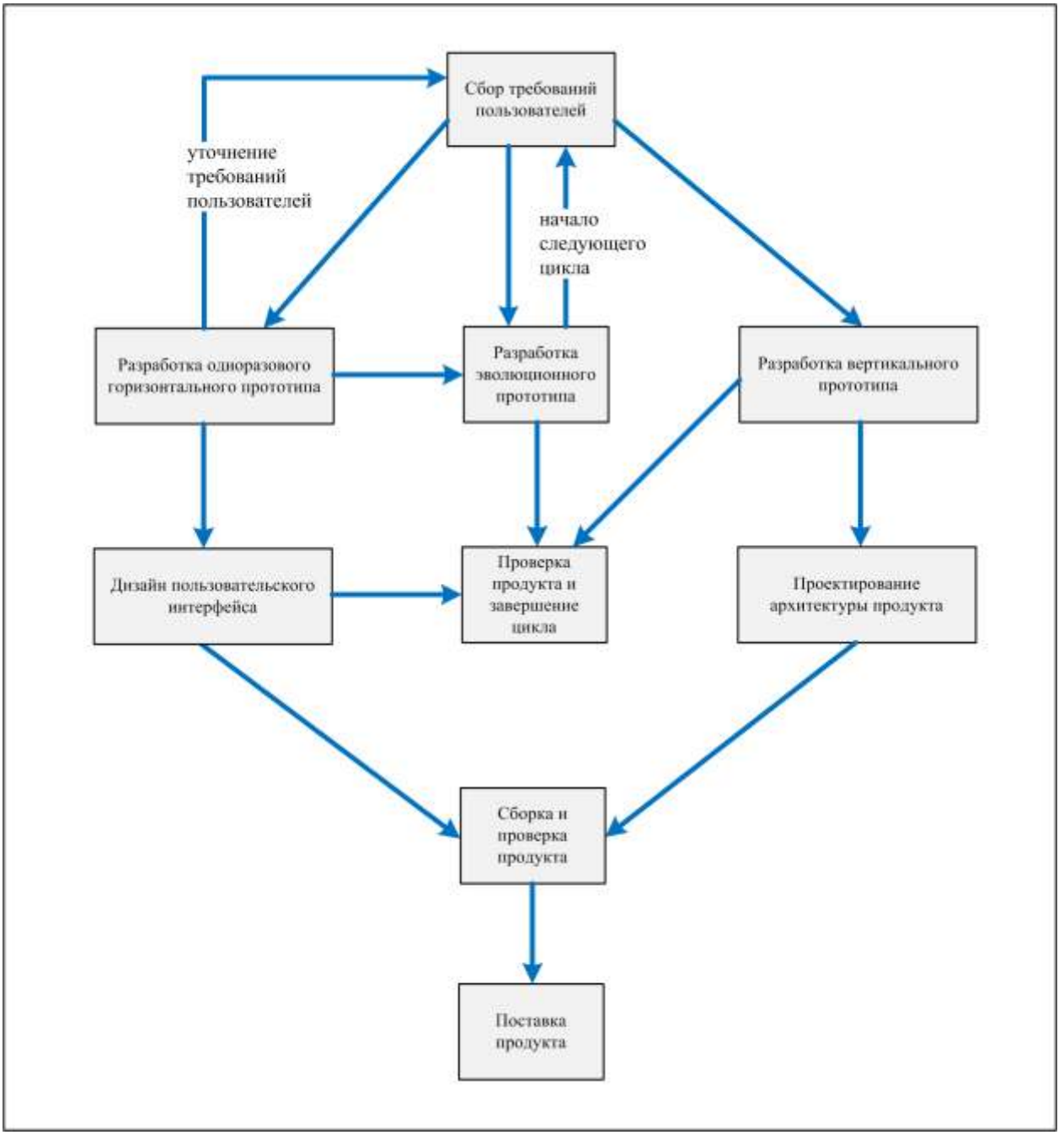


Рис. 13-2. Несколько возможных способов использования прототипов в процессе разработки ПО

Таблица 13-1. Стандартные способы применения прототипов ПО

	Одноразовые	**Эволюционные**
Горизонтальные	• Прояснение и уточнение примеров использования и функциональных требований • Выявление пропущенных: функций • Исследование возможных вариантов интерфейса пользователя	• Реализация базовых вариантов использования • Реализация дополнительных вариантов использования по приоритетам • Реализация и доработка Web-сайтов • Адаптация системы к быстро меняющимся требованиям бизнеса

Вертикальные	• Демонстрация технической осуществимости	• Реализация и наращивание ключевой клиент-серверной функциональности и уровней коммуникации • Реализация и оптимизация основных алгоритмов • Тестирование и настройка производительности

Бумажные и электронные прототипы

Не всегда для разрешения неопределенностей в требованиях нужен прототип в виде исполняемого кода. ***Бумажный прототип*** (paper prototype), иногда его называют ***низкокачественным прототипом,*** — это дешевый, быстрый и низкотехнологичный способ выяснить, как может выглядеть некий фрагмент системы (Rettig, 1994; Hohmann, 1997). Бумажные прототипы помогают установить, действительно ли пользователи и разработчики одинаково понимают требования. Они позволяют вам попробовать, практически не рискуя, сделать шаг при разработке кода продукта. Похожий метод, называемый ***методом раскадровки*** (storyboard) (Leffingwell и Widrig, 2000), показывает предлагаемый интерфейс пользователя, без привлечения пользователей к работе с ним.

Для создания бумажных прототипов требуются инструменты не сложнее, чем бумага, учетные карточки, наклейки и чистые пластиковые прозрачки. Проектировщик делает наброски экранов, как он их представляет, не заботясь о том, где точно будут располагаться элементы управления и как они будут выглядеть. Пользователи с готовностью делятся своим мнением, в результате чего возможны глубокие изменения — но на листе бумаги. В некоторых случаях, однако, они не склонны критиковать полюбившийся им компьютерный прототип, в который разработчик, по всей видимости, вложил много труда. Разработчики также иногда противятся внесению существенных изменений в тщательно выполненный электронный прототип.

При применении низкокачественного прототипа человек играет роль компьютера, пока длится оценочный сценарий. Он инициирует действия, произнося вслух, что он собирается сделать в конкретном экране: «Я хочу выбрать команду Предварительный просмотр из меню Файл». При этом пользователь показывает страницу или учетную карточку, представляющую экран, который должен появиться после выполнения действия. Это позволяет оценить, соответствует ли ответ ожидаемому и содержит ли экран необходимые элементы. Если нет, вы просто берете чистый лист бумаги или карточку и рисуете все заново.

Ищем волшебника

Команда разработчиков, создававшая копировальные устройства, однажды пожаловалась мне, что у их последнего устройства обнаружились проблемы с удобством использования. Самая обычная задача по копированию выполнялась в пять отдельных действий, что пользователям казалось ужасно неудобным. «Надо было нам сделать прототип этой задачи прежде, чем создавать продукт», — тоскливо сказал один из разработчиков.

Но как же прототипировать такой сложный продукт, как копировальный аппарат? Во-первых, купите большой телевизор. Во-вторых, напишите на коробке от него «КОПИРОВАЛЬНЫЙ АППАРАТ». Потом посадите кого-нибудь внутрь коробки и попросите пользователя подойти снаружи и изображать различные операции с аппаратом. Человек внутри коробки будет отвечать так, как, по его мнению, реагировало бы устройство, а представитель пользователей должен отмечать, та ли это реакция, которую он себе представляет. С помощью простого, веселого прототипа, подобного этому, — иногда его называют «прототипом волшебника из страны Оз» — зачастую удается на ранних стадиях разработки получить реакцию пользователей, руководствуясь которой разработчики и строят решения. Плюс вы получаете большой телевизор.

Какими бы совершенными ни были ваши инструменты прототипирования, наброски экранов на бумаге делать все равно быстрее. Бумажное прототипирование ускоряет каждый цикл, а это — ключевой фактор успеха при разработке требований. Это отличный метод уточнения требований для проектирования детализированных интерфейсов пользователей, создания эволюционных прототипов или традиционного проектирования и конструирования. Он также помогает команде лучше управлять ожиданиями клиентов.

Если вы решите создать электронный одноразовый прототип, воспользуйтесь соответствующими инструментами (Andriole, 1996). Некоторые из них перечислены здесь:

- языки программирования, такие, как Microsoft Visual Basic, IBMVisualAge Smalltalk и Inprise Delphi;
- языки подготовки сценариев, такие, как Perl, Python и Rexx;
- коммерческие наборы инструментальных средств прототипирования, средства раскраски изображений на экране и компоновщики
- графического интерфейса пользователя; средства рисования, такие, как Microsoft Visio и Microsoft Powe Point.

Средства для Web, использующие легко модифицируемые страницы HTML (Hypertext Markup Language), годятся для создания прототипов, которые предназначены для прояснения требований. Однако они не позволяют быстро исследовать детализированный дизайн интерфейсов. Соответствующие инструменты позволят вам с легкостью реализовать и обновлять компоненты интерфейса пользователя, вне зависимости от неэффективности его кода. Конечно же, если вы создаете эволюционный прототип, то должны использовать высококачественные средства разработки с самого начала.

Оценка прототипа

Для улучшения оценки горизонтальных прототипов создавайте сценарии, проводящие пользователей через последовательность действий, и задавайте конкретные вопросы, чтобы выявить требуемую информацию. Этот метод — дополнение к общему приглашению: «Расскажите, что вы думаете об этом прототипе». Сценарии оценки составляйте на основании вариантов использования или функций, которые должен представлять прототип. Сценарий попросит пользователей выполнить определенные задачи и проведет их через наиболее неясные

области прототипа. В конце выполнения каждой задачи и, возможно, в промежуточных пунктах, сценарий задает связанные с задачами вопросы. Ну и, конечно же, вы вправе задать вопросы сами.

- Реализует ли прототип все необходимые функции так, как вы этого ожидали?
- Не упущены ли какие-либо необходимые функции?
- Заметили ли вы какие-либо условия ошибки, не учтенные в прототипе?
- Нет ли в прототипе каких-либо ненужных функций?
- Насколько логичной и полной вам кажется навигация?
- Не оказалось ли выполнение каких-либо задач слишком сложным?

Удостоверьтесь, что пользователи оценивали прототип с соответствующих точек зрения. Привлекайте как опытных пользователей, так и новичков. Показывая прототип тем, кто его будет оценивать, подчеркните, что в нем воплощена лишь часть нужных функций, остальные будут реализованы в конечной системе.

Ловушка

Остерегайтесь пользователей, которые вводят реальные данные в оцениваемый прототип, потому что он им кажется реальной системой. Они будут недовольны, если по окончании работы с прототипом введенные ими данные будут потеряны.

Вы узнаете больше, наблюдая за работой пользователей с прототипом, чем просто опрашивая их. Формальное тестирование легкости и простоты использования эффективно, но простое наблюдение за пользователями в процессе также может многое сказать. Смотрите, к каким клавишам инстинктивно тянутся пальцы пользователя. Отмечайте места, где прототип конфликтует с другими приложениями, которыми часто пользуются клиенты, или где последующие действия пользователя не очевидны. Отмечайте нахмуренные брови, свидетельствующие, что пользователь озадачен и не знает, что делать дальше, как добраться до нужной точки или открыть другую часть приложения, чтобы просмотреть еще что-нибудь.

Просите ваших помощников вслух выражать их мнение во время работы с прототипом, чтобы вы могли понимать ход их мышления и отметить требования, которые прототип обрабатывает плохо. Создайте дружелюбную атмосферу, в которой пользователи будут свободно высказывать свои идеи и замечания. Избегайте показывать им «правильный» путь реализации тех или иных функций в прототипе.

Записывайте все, что узнаете в процессе оценки прототипа. Горизонтальный прототип позволит вам уточнить спецификацию требований к ПО. Если на основе оценки прототипа были приняты какие-либо решения о дизайне интерфейса пользователя или выбраны конкретные методы взаимодействия, запишите эти выводы и то, как вы к ним пришли. Иначе вам придется снова и снова возвращаться к одному и тому же — пустая трата времени. После вертикального прототипирования задокументируйте процесс и результаты оценки, в том числе решения, которые вы приняли об исследованных технических процессах. Ищите любые несоответствия между спецификацией требований к ПО и прототипом.

Риски прототипирования

Несмотря на то, что прототипирование уменьшает риск неудачи проекта по разработке ПО, оно обладает собственными рисками. Наибольший из них заключается в том, что кто-либо из заинтересованных в проекте лиц, увидев действующий прототип, решит, что окончательный продукт почти готов. «Ого, да вы, похоже, уже все сделали! — с энтузиазмом говорит тот, кого вы попросили оценить прототип.— Выглядит отлично. Может, вы быстро доделаете его и отдадите мне?»

Если коротко, то: **«НЕТ!»** Одноразовый прототип ни в коем случае не предназначен для работы, как бы он ни был похож на готовый продукт. Это всего лишь модель, образец,

эксперимент. Если нет жестких коммерческих обстоятельств, заставляющих немедленно застолбить место на рынке (в этом случае руководство понимает и принимает необходимость высоких затрат на сопровождение), сопротивляйтесь всем, кто настаивает на поставке одноразового прототипа в качестве готового продукта. Такой шаг в действительности лишь отдалит сроки завершения продукта, потому что и структура, и код одноразового прототипа намеренно создавались без учета качества или надежности.

Ловушка

Остерегайтесь тех заинтересованных в проекте лиц, которые думают, что прототип — это просто ранняя версия окончательного продукта. Управление ожиданиями — одна из ключевых составляющих успеха прототипирования. Каждый, кто увидит прототип, должен понимать его назначение и границы применения.

Не позволяйте страхам, связанным с преждевременным выпуском продукта, отвратить вас от создания прототипа, объясните всем, что вы не выпустите его как окончательный продукт. Один из способов контролировать этот риск — использовать бумажные, а не электронные прототипы. Ни одному из тех, кто оценивает бумажный прототип, и в голову не придет, что продукт уже почти готов. Еще один способ — выбрать инструменты прототипирования, отличающиеся от тех, что применяются для разработки окончательного продукта. Это поможет противостоять тем, кто просит «быстро закончить» и выпустить прототип. Оставив внешний вид прототипа недоделанным и незаконченным, вы также уменьшите этот риск.

Опасайтесь также, когда пользователей начинает мучить вопрос «Как?»: ***как*** интерфейс пользователя будет выглядеть и действовать. Работая с прототипами, внешне похожими на окончательный продукт пользователи легко забывают, что на стадии уточнения требований они должны в основном думать, ***что*** же они хотят видеть в системе. Ограничьте прототип только теми демонстрациями, функциями и возможностями навигации, которые помогут вам устранить неопределенности в требованиях.

Третий риск состоит в том, что пользователи начнут делать выводы о производительности конечного продукта по производительности прототипа. Вы не будете проводить оценку горизонтального прототипа в рабочей среде продукта. Возможно, для его создания вы использовали менее эффективные средства, например интерпретируемые сценарии, а не компилируемый код. В вертикальном прототипе не всегда применяются отлаженные алгоритмы или уровни защиты, что скажется на конечной производительности. Если пользователи увидят, что прототип мгновенно реагирует на моделируемый запрос к базе данных, используя жестко закодированные результаты запроса, они могут ждать такой же поразительной производительности от продукта, включающего огромную распределенную базу данных. Подумайте о реализации в прототипе временных задержек, чтобы модель ожидаемого поведения окончательного продукта выглядела более реалистичной (а прототип — менее готовым для реализации).

Наконец, опасайтесь действий по прототипированию, требующих таких усилий, что команда разработчиков выбьется из графика и будет вынуждена выпустить прототип в качестве готового продукта или торопливо и бессистемно доделывать продукт. Относитесь к прототипу, как к эксперименту. Вы проверяете, что требования достаточно определены, и что ключевые аспекты взаимодействия человека и компьютера, а также все архитектурные вопросы решены, так что можно переходить к проектированию и конструированию. Возможностей прототипа должно быть ровно сколько, сколько необходимо для проверки гипотез, ответов на вопросы и уточнения вашего понимания требований.

Факторы успеха прототипирования

Прототипирование ПО предлагает мощный набор методов, которые позволят сократить

сроки разработки, удовлетворить клиентов и создать продукты высокого качества. Чтобы сделать прототипирование эффективной частью процесса составления требований, прислушайтесь к следующим рекомендациям:

- включите задачи прототипирования в план вашего проекта. Составьте график распределения затрат времени и средств на разработку, оценку и модификацию прототипов;
- сформулируйте цель каждого прототипа до начала его создания;
- планируйте создание нескольких прототипов. В большинстве случаев вам не удастся создать то, что нужно, с первой попытки (в чем, собственно, и состоит весь смысл прототипирования!);
- одноразовые прототипы создавайте так быстро и дешево, как возможно. Вкладывайте минимум усилий в подготовку прототипов, которые помогут отвечать на вопросы или разрешать неясности в требованиях. Не доводите до совершенства одноразовые прототипы;
- не встраивайте в одноразовый прототип подробную проверку входных данных, методы безопасного программирования, процедур обработок ошибок или документации кода;
- не прототипируйте элементы, которые уже понимаете, кроме случаев исследования альтернатив дизайна;
- используйте правдоподобные данные в экранных формах и сообщениях прототипа. Пользователи будут отвлекаться и не смогут воспринять прототип как модель, описывающую внешний вид и поведение реальной системы;
- не ожидайте, что прототип полностью заменит спецификацию требований к ПО. Большая часть скрытой функциональности лишь подразумевается в прототипе и должна быть задокументировано в спецификации требований к ПО, чтобы ее удалось реализовать полно, точно и прозрачно. Видимая часть приложения — это пресловутая вершина айсберга. Изображения экранов не дают деталей определения полей данных, критериев проверки, взаимосвязей между полями (таких, как элементы управления пользовательского интерфейса, появляющихся, только если пользователь совершает определенные действия с другими элементами управления), обработки исключений, бизнес-правил и др.

Что теперь?

- Выделите часть вашего проекта, например один из вариантов использования, отражающий путаницу в требованиях. Набросайте часть возможного интерфейса, пользователя, который представляет ваше понимание требований и того, как они должны быть воплощены, — бумажный прототип. Попросите нескольких пользователей выполнить на вашем прототипе сценарий использования. Выявите места, где первоначальные требования были неполными или неверными. Модифицируйте прототип соответствующим образом и проведите повторный анализ, чтобы удостовериться, что недостатки устранены.
- Кратко перескажите содержание этой главы тем, кто будет оценивать ваш прототип, чтобы помочь им понять логику прототипирования и настроиться на реалистичные ожидания результатов.

Глава 14 Назначение приоритетов требований

После того как большинство требований к Chemical Tracking System было

задокументировано, менеджер проекта Дэйв и аналитик требований Лори встретились с двумя сторонниками продуктов. Тим представлял химиков, а Роксана выступала от лица работников склада химикатов.
«Как вы знаете, — начал Дэйв, — сторонники продуктов выдвинули массу требований к Chemical Tracking System. К сожалению, мы не можем включить всю запрашиваемую вами функциональность в первый выпуск. Поскольку авторы большинства требований - химики и работники склада, я хотел бы обсудить с вами расстановку приоритетов в требованиях».
Тим был озадачен. «Зачем вам назначать приоритеты? Все требования важны, иначе бы мы не дали их вам». Дэйв объяснил: «Я знаю, что они все важны, но мы не можем реализовать их все и при этом создать высококачественный продукт в заданные сроки. Мы хотим приложить все усилия, чтобы все самые необходимые требования вошли в первый выпуск, который должен появиться в конце следующего квартала. Мы просим вас помочь нам отделить требования, которые необходимо выполнить первыми, от тех, которые могут ждать». «Я знаю, что отчеты, которые составляет отдел охраны труда и техники безопасности, должны быть готовы к концу квартала или у компании будут неприятности, — заметила Роксана. — Мы можем использовать нашу нынешнюю систему учета еще несколько месяцев, если придется. Но распечатка и сканирование штрих-кодов нужны обязательно, это важнее, чем возможность поиска в каталогах поставщиков».
Тим запротестовал: «Я обещал химикам функцию поиска по каталогу в качестве примера того, как эта система будет экономить им время. Поиск по каталогу необходимо реализовать с самого начала».
Лори, аналитик, сказала: «При обсуждении с химиками вариантов использования мне показалось, что некоторые из них будут применяться очень часто, а другие — лишь время от времени и только одним-двумя людьми. Не могли бы мы взглянуть на ваш полный список вариантов использования и решить, какие из них не нужны вам немедленно? Также, я хотела бы отсрочить реализацию некоторых декоративных функций, указанных в приоритетных вариантах использования, если это возможно».
Тим и Роксана не особо взволновались от того, что придется подождать с реализацией некоторых функций системы. Тем не менее они поняли, что ожидать чудес неразумно. Если продукт не может содержать все требования в версии 1.0, то для всех будет лучше, если удастся согласовать первоочередные требования.

Для каждого продукта, на разработку которого выделены ограниченные ресурсы, необходимо определить относительные приоритеты возможностей. Расстановка приоритетов помогает менеджеру проекта разрешать конфликты, планировать выпуски и принимать необходимые компромиссы. В этой главе рассказано о важности определения приоритетов требований, предлагается простая схема классификации приоритетов и описан процесс точного анализа расстановки приоритетов, основанный на ценности, затратах и риске,

Зачем определять приоритеты требований

Когда ожидания клиентов высоки, а сроки поджимают, вам нужно, чтобы в продукте были реализованы самые ценные функции как можно раньше. Приоритеты — это способ разрешения борьбы между конкурирующими требованиями за ограниченные ресурсы. Определение относительного приоритета каждой возможности позволяет вам так планировать разработку, чтобы обеспечивать наибольшую ценность при наименьших затратах. Определение приоритетов наиболее критично для работы в очень строгих временных рамках или при применении инкрементальной модели с жесткими, фиксированными сроками выпуска каждой версии продукта. При экстремальном программировании клиенты выбирают, какие ***рассказы пользователей*** (user stories) они желают видеть в каждом двух- или трехнедельном выпуске, а разработчики оценивают, сколько из этих рассказов они могут вместить в каждый выпуск.

Менеджер проекта должен сбалансировать желаемый объем проекта и ограничения,

определяемые сроком, бюджетом, людскими ресурсами и качеством. Один из способов достижения этого — убрать (или отложить до более поздней версии) требования с низким приоритетом, когда принимаются новые, более важные требования или изменяются другие условия проекта. Если клиенты не классифицируют свои требования по важности и срочности, менеджерам проектов приходится делать это своими силами. Неудивительно, что клиентам не всегда нравится результат; поэтому они должны указывать, какие требования необходимы с самого начала, а какие могут подождать. Определяйте приоритеты на ранних стадиях проекта, когда больше шансов на успешный результат, и периодически возвращайтесь к ним.

Достаточно сложно заставить даже одного клиента решить, какие из его требований приоритетнее. Согласовать мнения нескольких клиентов с различными ожиданиями еще труднее. Люди по природе склонны отстаивать свои интересы и не жаждут жертвовать собственными нуждами ради чужой выгоды. Тем не менее, участие в определении приоритетов требований — одна из обязанностей клиента в отношениях «клиент - разработчик», как уже говорилось в главе 2. Обсуждение приоритетов помогает не только определить последовательность реализации требований, но и прояснять ожидания клиентов.

И клиенты, и разработчики должны внести вклад в определение приоритетов требований. Клиентам больше всего нужны функции, наиболее ценные для бизнеса или удобства работы. Однако, когда разработчики обрисуют затраты, трудоемкость, технический риск или компромиссы, связанные с каждым требованием, клиенты могут передумать и прийти к выводу, что это требование не так важно, как они считали изначально. Разработчики же иногда решают на ранних стадиях реализовать некоторые функции с низким приоритетом из-за их влияния на архитектуру системы,

Игры, в которые люди играют с приоритетами

Естественная реакция пользователей на просьбу определить приоритеты обычно такая: «Мне нужны все эти функции. Уж как-нибудь реализуйте их». Трудно убедить клиентов обсуждать приоритеты, если они знают, что функций с низким приоритетом не получат никогда. Один разработчик как-то сказал мне, что в их компании не принято назначать требованию низкий приоритет. Категории приоритетов требований назывались у них ***«высокие», «очень высокие»*** и ***«невероятно высокие»***. Другой разработчик заявлял, что назначение приоритетов совсем не нужно: если он записал что-либо в спецификацию требований, то собирается это реализовать. Но это не отвечает на вопрос, ***когда*** будет реализована каждая функция. Некоторые разработчики избегают определения приоритетов, потому что оно не соответствует позиции «мы можем все», которую они декларируют.

Ловушка

Избегайте определения приоритетов «по децибелам», когда требование, высказанное громче всего, получает наибольший приоритет, и «по угрозе», когда те, кто имеют больший вес в компании, всегда получают требуемое.

В действительности некоторые возможности системы более ценны, чем остальные. Это становится очевидным, когда на поздних стадиях требуется «быстро урезать» проект, то есть отбросить несущественные характеристики, чтобы критически важные возможности удалось реализовать в срок. Определив приоритеты на ранних этапах разработки проекта и переоценивая их в соответствии с изменяющимися предпочтениями клиентов, условиями рынка и реалиями бизнеса, команда сможет мудро тратить свое время на создание наиболее ценных возможностей. Реализовав большую часть функции, а затем решив, что она не нужна, вы впустую потратите ресурсы и испытаете горькое разочарование.

Предоставленные самим себе, клиенты, скорее всего назначат 85% требований высокий приоритет, 10% — средний и 5% — низкий. Это не дает менеджеру проекта большой свободы для

маневра. Если почти все требования действительно важны, вы рискуете тем, что не весь ваш проект будет успешен, поэтому придется составить планы соответствующим образом. Отшлифуйте требования до блеска, чтобы отбросить не имеющие большой ценности и упростить неоправданно сложные требования (McConnell, 1996). Чтобы представители клиентов с большим мужеством назначали требованиям низкие приоритеты, аналитику стоит задать вопросы, подобные перечисленным ниже.

- Есть ли другой способ удовлетворить это требование клиентов?
- Что случится, если это требование убрать или отложить?
- Что произойдет с бизнес-целями, если это требование не будет реализовано немедленно?
- Почему пользователи будут недовольны, если реализацию этого требования отложить до следующего выпуска?

Однажды команда управления менеджментом в крупном коммерческом проекте выразила нетерпение, когда аналитик настаивал на определении приоритетов требований. Менеджеры отметили, что зачастую они могут обойтись без какой-либо функции, но тогда в компенсацию придется усиливать другую функцию. Если бы они отсрочили слишком много функций, то продукт не принес бы запланированные доходов. Оценивая приоритеты, учитывайте связи и взаимосвязи между различными требованиями, а также их соответствие бизнес-целям проекта.

Шкала приоритетов

Обычный метод расстановки приоритетов подразумевает три группы категорий требований. Неважно, как вы назовете их, все сводится *к.* высокому, среднему и низкому приоритетам. Такие шкалы приоритетов субъективны и неточны. Лица, заинтересованные в проекте, должны согласовать, что означает каждый уровень в используемой шкале.

Один из способов оценки приоритетов предлагает учитывать два измерения: ***важность*** и ***срочность*** (Covey, 1989). Каждое требование считается важным либо не важным и срочным либо не срочным. Как показано в табл. 14-1, получаются четыре комбинации для определения шкалы приоритетов:

- требования с ***высоким приоритетом*** (high priority) — и важные (пользователям нужны функции), и срочные (они необходимы уже в следующем выпуске). Некоторые требования приходится включать в эту категорию согласно контрактным или юридическим обязательствам либо из-за непреодолимых бизнес-причин;
- требования со ***средним приоритетом*** (medium priority) — важные (пользователям нужны функции), но не срочные (они могут ждать следующего выпуска);
- требования с ***низким приоритетом*** (low priority) — не важные (пользователи при необходимости могут обойтись без этой функций) и не срочные (пользователи могут ждать, причем вечно);
- требования в четвертой клетке кажутся срочными, но в действительности — они не важны. Не тратьте время на работу над ними. Они не сделают продукт более ценным.

Таблица 14-1. Определение приоритетов требований по важности и срочности

	Важные	**Не важные**
Срочные	Высокий приоритет	Не занимайтесь ими!
Не срочные	Средний приоритет	Низкий приоритет

Включайте приоритеты каждого требования в описания вариантов использования, спецификацию требований или базу данных требований. Установите правила, чтобы пользователь знал, присваивается ли приоритет, назначенный высокоуровневому требованию, всем дочерним требованиям или же каждое функциональное требование должно иметь свой собственный атрибут приоритетности.

Даже проект средних масштабов может иметь сотни функциональных требований — слишком много, чтобы классифицировать их аналитически и последовательно. Чтобы не потерять контроль над ситуацией, выберите для определения приоритетов определенный уровень абстракции — возможности, варианты использования или функциональные требования. В рамках одного варианта использования некоторые альтернативные направления могут иметь больший приоритет, чем остальные. Вы имеете возможность сначала определить приоритеты на уровне функций, а затем назначить приоритеты функциональным требованиям отдельно для каждой функции. Это поможет вам отделить основную функциональность от усовершенствований, которые предполагается реализовать позже. Документируйте даже требования с низким приоритетом. Их приоритет может впоследствии измениться, и если разработчики знают о них сейчас, им проще планировать будущую модификацию.

Определение приоритетов на основе ценности, стоимости и риска

В малом проекте заинтересованные лица могут согласовать приоритеты требований неформально. Крупные или спорные проекты требуют более структурированного подхода, устраняющего из процесса некоторые эмоции, политику и догадки. Для помощи в определении приоритетов предлагается несколько аналитических и математических методик. Они предполагают оценку относительной ценности и относительной стоимости каждого требования. Требования с самым высоким приоритетом — те, что обеспечивают большую ценность продукта при меньшей стоимости (Karlsson и Ryan, 1997; Jung, 1998). Субъективная оценка стоимости и ценности посредством попарного сравнения всех требований не годится, если требований более двух дюжин.

Другая альтернатива— Quality Function Deployment (QFD), всесторонний метод определения относительной ценности для клиента предлагаемых функций продукта (Zultner, 1993; Cohen, 1995). Третий подход, заимствованный из Total Quality Management (TQM), позволяет оценить каждое требование по нескольким весомым критериям успеха проекта и подсчитать количество баллов для назначения приоритетов требований. Тем не менее, по-видимому, лишь немногие организации-разработчики ПО готовы применять QFD или TQM.

В табл. 14-2 показана крупноформатная модель, которая поможет вам оценить относительные приоритеты для набора вариантов использования, функций или функциональных требований. Загрузить эту таблицу в формате Microsoft Excel можно с *http://www.processim-pact.com/goodies.shtml*. В табл. 14-2 перечислено несколько функций Chemical Tracking System (а каких же еще?). Эта схема заимствует и QFD концепцию обоснования ценности для пользователя; учитывается как выгода для пользователя, если функция реализована в продукте, так и неудобство, если она отсутствует (Pardee. 1996). Привлекательность функции прямо пропорциональна ее полезному действию и обратно пропорциональна стоимости и риску, связанному с ее реализацией. При прочих равных условиях функции с наибольшим значением ценность/стоимость должны иметь наивысший приоритет. При этом набор оцениваемых приоритетов распределяется по всему континууму, а не по нескольким отдельным уровням.

Примените эту схему определения приоритетов к дискреционным требованиям, не имеющим наивысшей ценности. Не нужно включать в этот анализ элементы, которые реализуют основные бизнес-функции продукта, которые вы считаете основными отличительными чертами продукта или которые необходимы для соответствия юридическим нормам. Если нет возможности назначить этим функциям со временем более низкий приоритет при изменении условий, то необходимо реализовать их в продукте как можно быстрее. Определив функции, которыми обязательно должен обладать выпускаемый продукт, используйте табл. 14-2, чтобы оценить по шкале относительных приоритетов оставшиеся возможности. Обычно в процессе назначения приоритетов участвуют:

- менеджер проекта, который ведет процесс, разрешает конфликты и при необходимости адаптирует данные, поступающие от других: участников;
- представители клиента — сторонники продукта или маркетологи, предоставляющие информацию о сильных и слабых сторонах продукта;

- представители разработчиков, например руководители команд, сообщающие данные о стоимости и риске.

Таблица 14-2. Образец матрицы определения приоритетов для Chemical Tracking System

Относительный вес	**2**	**1**			**1**		**0,5**		
функция	**Относительная выгода**	**Относительный урон**	**Общая ценность**	**Ценность в %**	**Относительная стоимость**	**Стоимость в %**	**Относительный риск**	**Риск в %**	**Приоритет**
1.Распечатка списка данных по безопасности материалов	**2**	**4**	**8**	**5,2**	**1**	**2,7**	**1**	**3,0**	**1,22**
2.Запрос о статусе заказа поставщика	**5**	**3**	**13**	**8,4**	**2**	**5,4**	**1**	**3,0**	**1,21**
3.Создание отчета о инвентаризации склада химикатов	**9**	**7**	**25**	**16,1**	**5**	**13,5**	**3**	**9,1**	**0,89**
4.Просмотр истории использования каждого контейнера с химикатом	**5**	**5**	**15**	**9,7**	**3**	**8,1**	**2**	**6,1**	**0,87**
5.Поиск химиката по каталогам поставщиков	**9**	**8**	**26**	**16,8**	**3**	**8,1**	**8**	**24,2**	**0,83**
6.Поддержка списка опасных химикатов	**3**	**9**	**15**	**9,7**	**3**	**8,1**	**4**	**12,1**	**0,68**
7.Модификация невыполненных заявок на химикаты	**4**	**3**	**11**	**7,1**	**3**	**8,1**	**2**	**6,1**	**0,64**
8.Создание отчетов об инвентаризации отдельных лабораторий	**6**	**2**	**14**	**9,0**	**4**	**10,8**	**3**	**9,1**	**0,59**
9.Проверка в базе данных по обучению наличия записи о прохождении обучения по обращению с опасными веществами	**3**	**4**	**10**	**6,5**	**4**	**10,8**	**2**	**6,1**	**0,47**
10.Импорт химических структур из инструментальных средств для рисования структур	**7**	**4**	**18**	**11,6**	**9**	**24,3**	**7**	**21,2**	**0,33**
Итоги	**53**	**49**	**155**	**100,0**	**37**	**100,0**	**33**	**100,0**	

При использовании этой модели определения приоритетов пользуйтесь следующим планом.

1. Перечислите в таблице все функции, варианты использования или требования, для которых хотите определить приоритеты; я для примера взял функции. Все элементы должны принадлежать к одному уровню абстракции — не смешивайте функциональные требования с функциями продукта. Если определенные функции логически связаны (например, вы реализуете функцию В только при наличии функции А), в анализ включайте только ведущую функцию. Эта модель работает лишь для нескольких дюжин функций, далее она становится слишком громоздкой. Если у вас несколько дюжин элементов, группируйте связанные функции, чтобы составить управляемый первоначальный список. При необходимости вы можете провести второй раунд анализа, на более детальном уровне.

2. Попросите представителей клиентов оценить относительную выгоду, которую каждая функция дает клиенту или бизнесу, по шкале от 1 до 9: 1 балл означает, что никто не находит ее полезной, а 9 — что она крайне ценная. Эти оценки выгоды отражают связь функций с бизнес-требованиями к продукту.

3. Оцените относительный урон, который потерпит клиент или бизнес, если функция не будет включена в продукт. Снова используйте шкалу от 1 до 9: 1 балл означает, что никто не расстроится, если ее не окажется; 9 показывает серьезный урон. Требования, имеющие низкие рейтинги и выгоды, и урона, увеличивают стоимость, но имеют малую ценность; они могут оказаться мишурой: выглядят привлекательно, но не стоят инвестиций. Оценивая урон, учитывайте, насколько расстроятся клиенты, если определенная функция не будет реализована. Задайте себе вопросы, аналогичные перечисленным ниже.

- Проиграет ли ваш продукт по сравнению с другими, содержащими эту функцию?
- Будут ли какие-либо юридические или контрактные последствия?
- Не нарушите ли вы какой-либо юридический или промышленный стандарт?
- Не лишит ли это пользователей возможности выполнять какие-либо необходимые или ожидаемые действия?
- Намного ли сложнее добавить эту функцию позже, в качестве модернизации?
- Возникнут ли проблемы из-за того, что отдел маркетинга обещал эту функцию, чтобы удовлетворить некоторых потенциальных клиентов, но команда решила не включать ее?

4. В этой таблице подсчитаны итоговые значения для каждой функции как сумма баллов ее выгоды и урона. По умолчанию выгода и урон оцениваются одинаково. Для уточнения вы можете изменить относительный вес этих двух параметров в верхней строке таблицы. В примере, взятом для табл. 14-2, выгода оценивается дважды, как и урон. В таблице суммируется ценность всех функций и посчитан процент от общего значения для каждого набора функций, равного сумме ценностей каждой функции (%).

5. Попросите разработчиков оценить относительную стоимость реализации каждой функции, опять-таки и по шкале от 1 (легко и быстро) до 9 (трудоемко и дорого). В таблице подсчитан процент общей стоимости, составленной из вклада каждой функции. Разработчики оценивают рейтинги стоимости, учитывая сложность функции, объем требуемой работы над интерфейсом пользователя, потенциальную возможность повторного использования существующего кода, объем необходимого тестирования и документации и т.д.

6. Подобным же образом, разработчики оценивают относительную степень технического или другого риска, связанного с каждой функцией, по шкале от 1 до 9. Технический риск - это вероятность неудачной реализации функции с первого раза. 1 балл означает, что вы сможете запрограммировать ее даже во сне, 9 баллов означает серьезную озабоченность возможностью реализации, нехваткой сотрудников с необходимым опытом или использованием неиспытанных или незнакомых средств и технологий. Плохо определенные требования, которые могут нуждаться в переработке, следуют из высоких оценок риска. В таблице подсчитан общий процент риска, который складывается из риска каждой функции.

В стандартной модели выгода, урон, стоимость и риск оцениваются одинаково, но вы можете изменить их вес. В табл. 14-2 риск имеет половину веса фактора стоимости, который имеет тот же

вес, что и урон. Если вы не желаете учитывать риск вообще в этой модели, приравняйте значение его веса к нулю.

7. После того как вы введете все результаты оценки в таблицу, по следующей формуле подсчитывается значение приоритета для каждой функции:

$$\text{приоритет} = \frac{\text{ценность\_\%}}{(\text{стоимость\_\%} \times \text{вес\_стоимости}) + (\text{риск\_\%} \times \text{вес\_риска})}$$

8. Отсортируйте список функций по уменьшению подсчитанного приоритета. Функции вверху списка характеризуются наиболее благоприятным сочетанием ценности, стоимости и риска и поэтому — при равенстве остальных факторов — должны иметь наивысший приоритет.

А можно еще на кулачках

Одна компания, которая ввела у себя процедуру определения приоритетов, основанную на таблице, показанной в этой главе, обнаружила, что это помогло команде, работающей над проектом, выйти из тупика. Несколько заинтересованных в проекте лиц придерживались различных мнений о том, какие функции наиболее важны для проекта, и команда оказалась в безвыходном положении. Анализ с помощью таблицы сделал оценку приоритетов более объективной и менее эмоционально накаленной, позволив команде прийти к некоторым удовлетворительным заключениям и двигаться дальше.

Консультант Johanna Rothman (2000) писала: «я предлагала эту электронную таблицу своим клиентам в качестве инструмента для принятия решений. Хотя те, кто пробовал ей пользоваться, ни разу не заполнили ее до конца, они обнаружили, что дискуссия, которая начиналась при ее использовании, чрезвычайно полезна для определения относительных приоритетов различных требований». Таким образом, используйте концепцию выгоды, урона, стоимости и риска для направления дискуссий о приоритетах. Это более ценно, чем формально выполнить полную проработку анализа электронной таблицы и затем полагаться исключительно на вычисленную последовательность приоритетов.

Точность этого метода ограничена возможностью команды оценивать выгоду, урон, стоимость и риск для каждого элемента. Поэтому используйте подсчитанную последовательность приоритетов только как ориентир. Представители клиентов и разработчиков должны проверить итоговую таблицу, чтобы прийти к соглашению о рейтингах и результирующей последовательности приоритетов. Настройте эту модель под себя, используя набор готовых требований из какого-нибудь предыдущего проекта. Измените значения веса так, чтобы подсчитанная последовательность приоритетов хорошо сочеталась с вашей постфактум-оценкой того, насколько в действительности важны требования в вашем калибровочном наборе. Это даст вам некоторую уверенность в использовании данной таблицы в качестве модели для расстановки приоритетов в ваших проектах.

Ловушка

Не придавайте слишком большого значения малым отличиям в подсчитанных значениях приоритетов. Данный полуколичественный метод не претендует на математическую точность. Ищите группы требований с похожими значениями приоритетов.

Заинтересованные в проекте лица часто по-разному оценивают важность того или иного требования и потенциальный уровень от его отсутствия. Один из вариантов таблицы приоритетов учитывает входные данные от нескольких классов пользователей или других заинтересованных в проекте лиц. На странице ***Multiple Stakeholders*** (Множество пользователей) загружаемой электронной таблицы скопируйте колонки ***Relative Benefit*** (Относительная выгода) и ***Relative Penalty*** (Относительный урон) для каждого заинтересованного в проекте лица, участвующего в анализе. Затем назначьте вес для каждого участника, больший вес отдавая привилегированным классам пользователей, а не группам, имеющим меньшее влияние на принятие решений, касающихся проекта. В таблице будет учтен вес каждого участника при подсчете общего значения ценности.

Эта модель также поможет вам принимать решения о компромиссах при оценке предлагаемых изменений в требованиях. Учитывайте, как их приоритеты соответствуют приоритетам требований в принятой базовой версии, чтобы выбирать наилучшую последовательность реализации.

Старайтесь, чтобы ваш процесс определения приоритетов был насколько возможно простым, но не проще этого. Стремитесь превратить обсуждение требований из политической баталии в форум, где заинтересованные в проекте лица смогут давать честные оценки. Это даст вам лучшую возможность создавать продукты, обладающие максимальной бизнес-ценностью.

Что дальше?

- Примените модель определения приоритетов, описанную в этой главе, к 10-15 функциям или вариантам использования из недавнего проекта. Насколько хорошо подсчитанные этим способом приоритеты совпадают с теми, которые вы определили каким-либо другим методом? Насколько хорошо они соответствуют вашим субъективным ощущениям правильной расстановки приоритетов?
- Если обнаружилась диспропорция между приоритетами, предсказанными моделью, и теми, которые кажутся вам правильными, проанализируйте, какая часть модели не дает ожидаемых результатов. Попытайтесь использовать разные значения веса для выгоды, урона, стоимости и риска. Изменяйте модель до тех пор, пока она не даст результаты, соответствующие вашим ожиданиям. Настроив и изменив модель расстановки приоритетов, примените ее к новому проекту. Включите подсчитанные приоритеты в процесс принятия решений. Посмотрите, дает ли это результаты, которые заинтересованным в проекте лицам кажутся более удовлетворительными, чем те, что получались при прежнем подходе.

Глава 15 Утверждение требований

Барри, руководитель по тестированию, вел семинар, участники которого проверяли спецификацию требований к ПО на наличие проблем. Во встрече принимали участие представители двух самых крупных классов пользователей, разработчик Джереми и аналитик Триш, которая написала спецификацию требований к ПО. В требовании 83 было указано: "Система обеспечит безопасность рабочих станций, подключенных к DMV, посредством отключения неактивных терминалов по истечении тайм-аута». Джереми представил группе свою интерпретацию этого требования: В требовании 83 говорится о том, что система автоматически отключит пользователя любой рабочей станции, подключенной к системе DMV, если на ней наблюдается бездействие в течение определенного периода времени».

«У кого-нибудь есть вопросы по этому требованию?» - поинтересовался Барри.

Первым высказался Хью-Ли, один из сторонников продукта. «Как система определяет, что терминал не активен? Это как появление заставки на мониторе: если в течение, ну, скажем, пяти минут мышь или клавиатура остаются в покое, то сеанс пользователя закрывается? Это может раздражать, например, когда пользователь просто отвлекся на разговор в течение этих пяти минут».

Триш добавила: «На самом деле в требовании ничего не говорится об отключении пользователя. Я не совсем понимаю, что означает "безопасность... по истечении тайм-аута». Я предположила, что это означает окончание сеанса, но, может, достаточно просто ввести пароль».

Джереми был также озадачен. «В этом требовании речь идет о любой рабочей станции, которая может подключиться к системе DMV или о рабочих станциях, которые активны и подключены к системе DMV в данный момент? О времени ожидания какой длины мы говорим? Возможно, существует некое руководство по безопасности для таких случаев?» Барри убедился, что секретарь точно зафиксировал все точки зрения. «Ну хорошо, похоже, что в требовании 83 есть несколько неясностей и не хватает некоторой информации, в частности, не определено время ожидания. Триш, не могла ли бы ты связаться с координатором отдела безопасности и решить этот вопрос?».

Большинству разработчиков знакомо чувство расстройства, возникающее, если их просят реализовать слишком неясные или неполные требования. Если они не могут получить необходимую информацию, они вынуждены ориентироваться на собственные предположения, которые не всегда верны. Потребуется много усилий, чтобы исправить ошибки в требованиях, работа над которыми уже завершена. Исследования показали, что в 100 раз дороже исправлять ошибки в требованиях, если на них указывает клиент, чем в процессе разработки этих требований (Boehm, 1981; Grady, 1999). Другие исследования свидетельствуют, что на исправление ошибки, выявленной на стадии работы над требованиями, тратится в среднем 30 минут, тогда как на исправление ошибки, выявленной в ходе тестирования системы, необходимо от 5 до 17 часов (Kelly, Sherif и Hops, 1992). Ясно, что любые усилия, затраченные на выявление ошибок в спецификации к требованиям, сэкономят реальные время и деньги.

Во многих проектах тестирование — одна из последних стадий проекта. Проблемы, связанные с требованиями, могут оставаться в продукте до тех пор, пока они не будут выявлены в ходе долгого тестирования системы или их не обнаружит клиент. Если вы начнете планирование тестирования и разработку вариантов тестирования на ранней стадии проекта, вы сможете обнаружить многие ошибки вскоре после их появления. При этом вы предотвратите дополнительный ущерб, наносимый ими, и снизите затраты на тестирование и обслуживание.

На рис. 15-1 показана V-образная модель разработки ПО, когда действия, связанные с тестированием и разработкой проекта выполняются параллельно (Davis, 1993). Эта модель указывает, что приемочные испытания основывается на требованиях пользователей, тестирование

системы — на функциональных требованиях, а тестирование целостности — на архитектуре.

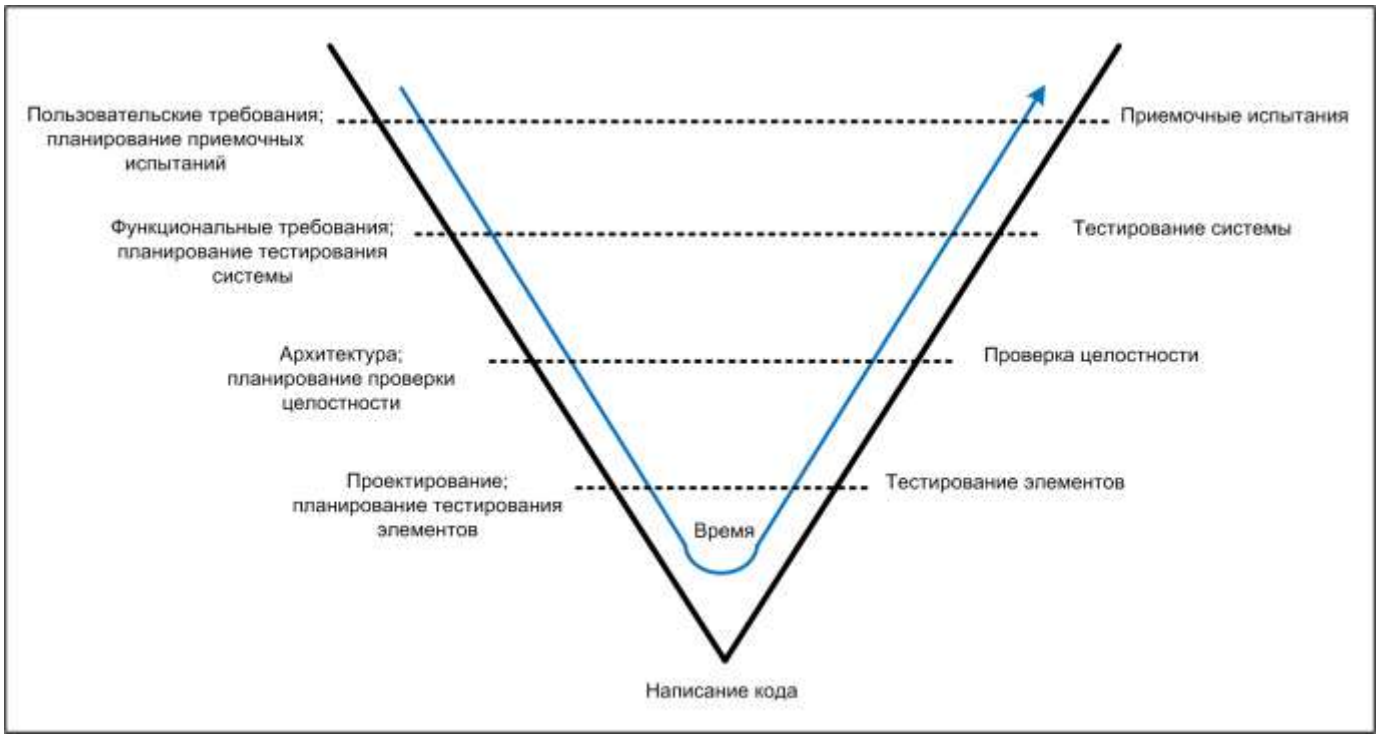


Рис. 15-1. V-образная модель разработки ПО подразумевает планирование и разработку тестирования на ранней стадии проекта

Планируйте мероприятия по тестированию и начинайте разработку предварительных вариантов тестирования для соответствующего этапа разработки. В процессе разработки требований вы не можете проводить никаких тестов, так как никакого ПО еще нет. Однако концептуальные (то есть, не зависящие от реализации) варианты тестировании, созданные на основе требований, позволят выявить ошибки, неясности и пропуски данных в спецификации требований к ПО и проанализировать модели задолго до того, как разработчики приступят к написанию кода.

Утверждение требований является четвертым этапом — наряду со сбором информации, анализом и созданием спецификации — разработки требований (Abran и Moore, 2001). [3] Утверждение требований позволяет удостовериться в том, что:

- в спецификации требований к ПО должным образом описаны предполагаемые возможности и характеристики системы, которые удовлетворят потребности различных заинтересованных в проекте лиц;
- требования к ПО точно отражают системные требования, бизнес-правила и др.;
- требования полные и высококачественные;
- все требования согласованы друг с другом;
- требования обеспечивают качественную основу для дизайна и сборки ПО.

Проверка подтверждает, что в документе представлены желаемые характеристики отличных требований (полные, корректные, осуществимые, необходимые, приоритетные, ясные и поддающиеся проверке) и отличных спецификаций к требованиям (полные, согласованные,

[3] Для описания этого этапа некоторые авторы используют термин *«проверка»* (Thayer и Dotfman, 1997). При проверке определяется, соответствует ли результат разработки на данном этапе требованиям к нему (делается ли все правильно). При утверждении оценивается, действительно ли продукт отвечает потребностям клиента (делается ли все в нужном направлении). В этой книге я использую терминологию «Software Engineering Body of Knowledge» (Abran и Moore, 2001) и называю четвертую составляющую разработки требования утверждением.

модифицируемые и поддающиеся отслеживанию). Разумеется, вы можете утвердить только задокументированные требования, а не предполагаемые, которые существуют только в чьем-то воображении.

Утверждение — это не один отдельный этап процесса, выполняемый после сбора и документирования требований. Некоторые проверочные мероприятия, например просмотр все более разрастающейся спецификации требований к ПО, выполняются после каждой процедуры сбора информации, ее анализа и документирования. Другие мероприятия, такие, как официальная проверка спецификации требований к ПО, обеспечивают достижение того уровня качества, которое предшествует финальной версии спецификации. Включите в план проекта этапы утверждения требований в качестве отдельных задач,

Участники проекта иногда с неохотой тратят время на проверку и тестирование спецификации требований к ПО. Интуитивно кажется, что если выделить время на улучшение качества требований, дата выпуска продукта задержится на такой же срок. Однако при таком отношении вы можете смело ожидать нулевых результатов от всех усилий, затраченных на утверждение. В действительности же эти усилия могут сократить график поставки, за счет уменьшения требуемых исправлений и ускорения интеграции и тестирования системы (Blackburn, Scudder и Van Wassenhove, 1996). Capers Jones (1994) отмечает, что каждый доллар, который вы истратили на предотвращение появления дефектов, снизит затраты на исправление на сумму от 3 до 10 долларов. Чем лучше требования, тем выше качество продукта и тем более доволен клиент, что в свою очередь снизит затраты на обслуживание, улучшение и клиентскую поддержку продукта. Инвестируя в качество продукта, вы сохраните больше денег, чем потратили.

Для оценки корректности и качества требований применяйте различные приемы (Wallace и Ippolito, 1997). Один из таких приемов заключается в измерении каждого требования, с тем чтобы вы смогли продумать способ измерения того, насколько хорошо предложенное решение. Suzanne и James Robertson используют термин ***критерий годности*** (fit criteria) для подобных приемов (Robertson и Robertson, 1999). В этой главе рассматриваются приемы проверки официальных и неофициальных просмотров требований, разработки вариантов тестирования на основании требований и определения клиентом критериев приемлемости продукта.

Просмотр требований

Всякое исследование продукта ПО на предмет выявления проблем любым другим лицом, кроме его автора, называется ***экспертной оценкой*** (peer review). Просмотр задокументированных требований — это эффективный прием выявления неясных или не поддающихся проверке требований, требований, которые были определены недостаточно ясно для разработки, а также других проблем.

Различные типы экспертных оценок называются по-разному (Wiegers, 2002a). Неофициальные просмотры полезны при знакомстве людей с продуктом и сборе отзывов о нем (формирование обратной связи). Однако они несистематические, неполные и несогласованные. Есть несколько видов неофициальных просмотров требований:

- ***проверка «за столом»,*** когда вы просите одного коллегу исследовать ваш продукт;
- ***коллективная проверка,*** когда вы приглашаете несколько коллег для параллельной проверки продукта;
- ***критический анализ,*** когда автор описывает создаваемый продукт и просит его прокомментировать.

В отличие от нарочито естественных неформальных просмотров, официальная проверка представляет собой строго регламентированный процесс. По его завершении формируется отчет, в котором указаны материал, рецензенты и мнение команды рецензентов о приемлемости продукта. Главный результат — совокупность всех найденных дефектов и поднятых вопросов. Члены команды, которая выполняет официальный просмотр продукта, делят ответственность за качество проверки, хотя в конце концов за качество продукта отвечают все-таки его создатели (Freedman и Weinberg, 1990).

Наиболее хорошо себя зарекомендовавшая себя форма официальной оценки называется ***экспертизой*** (Gilb и Graham, 1993; Radice, 2002). Возможно, инспектирование документации

требований — наиболее действенный из доступных приемов проверки качества ПО. Несколько компаний сэкономили ни много ни мало — целых 10 часов труда на каждый час, потраченный на инспектирование документации требований и других готовых к поставке продуктов (Grady и Van Slack, 1994). Инвестиции в проверки вернутся 1000% выгоды, если вы, конечно, не отнесетесь к ним небрежно.

Если вы серьезно решили совершенствовать качество вашего ПО, то ваша команда должна проверить каждый написанный документ, касающийся требований. Детальная проверка объемной документации может быть скучной и долгой. Однако все мои знакомые, кто занимался экспертизой требований, соглашаются, что каждая минута была потрачена не зря. Если у вас не хватает времени на проверку каждой детали, воспользуйтесь рискованными методами анализа: выберите требования, для проверки которых достаточно неофициального просмотра. Начинайте экспертизу спецификации требований к ПО, когда требования разработаны еще примерно на 10%. Выявление значительных дефектов на ранней стадии и систематических проблем в процессе написания требований является отличным способом предотвращения — а не только выявления — ошибок.

Чем тщательнее присматриваешься, тем больше замечаешь

В Chemical Tracking System представители пользователей проводили неофициальный просмотр последней версии спецификации требований после каждого семинара, посвященного сбору информации. Таким образом удавалось выявить множество ошибок. После завершения сбора информации, один аналитик свел всю информацию, полученную от всех классов пользователей в одну спецификацию требований к ПО (50 страниц плюс несколько приложений). Затем два аналитика, один разработчик, три сторонника продукта, менеджер проекта и один тестировщик проверили этот документ за три двухчасовых совещания, проведенных в течение недели. Они обнаружили 233 дополнительные ошибки, в том числе несколько серьезных дефектов. Все проверяющие согласились, что время, потраченное на «прочесывание» спецификации (по одному требованию за раз) сэкономило несчетное количество часов в дальнейшем.

Проведение экспертизы

Экспертиза была разработана Майклом Фэганом (Michael Fagan) из IIBM в середине 70-х гг. (Fagan, 1976), а другие дополнили и модифицировали его методы (Gilb и Graham, 1993). Экспертиза была признана лучшим приемом в области разработки ПО (Brown, 1996). Этот метод, годится для проверки любого продукта, в том числе документации требований и дизайна, исходного кода, документации тестирования и планов проекта.

Экспертиза — это четко определенный многоэтапный процесс. В нем участвует небольшая команда квалифицированных специалистов, которые тщательно проверяют продукт на наличие дефектов и изучают возможности улучшения. Экспертиза обеспечивает уровень качества продукта, предшествующий окончательному. Иногда возникают дебаты, действительно ли метод Фэгана можно считать лучшей формой проверки (Glass, 1999), однако ни у кого не возникает сомнений, что экспертиза является мощным приемом повышения качества продукта. Множество полезных советов по проведению экспертной оценки и экспертизы опубликовано на *http://www.processimpact.com/goodies.shtml, в* том числе описание процесса экспертной оценки, контрольные списки дефектов и формы экспертизы.

Участники

Участники инспектирования должны отражать четыре точки зрения на продукт (Wiegers, 2002a).

- **Автор продукта и, возможно, коллеги автора.** Это может быть аналитик, составивший документацию требований.
- **Автор любого предыдущего продукта или спецификации для элемента, который следует проверять.** В этой роли может выступить системный инженер или архитектор, который способен подтвердить, что в спецификации требований к ПО система описана достаточно детально. При отсутствии спецификации более высокого уровня в экспертизе должны принять участие представители клиента для того, чтобы подтвердить, что требования в спецификации требований к ПО описаны правильно и полно.
- **Люди, которые будут выполнять работу, основанную на проверяемом элементе.** Для экспертизы спецификации требований к ПО вы можете привлечь разработчика, тестировщика, менеджера проекта, а также составителя пользовательской документации. Они будут выявлять наличие проблем различных типов. Требование, которое не поддается проверке, скорее всего сможет выявить тестировщик, а разработчик сумеет определить требования, которые технически невыполнимы.
- **Люди, отвечающие за работу продуктов, взаимодействующих с проверяемым элементом.** Они выявят проблемы с требованиями к внешнему интерфейсу. Они также могут выявить требования, изменение которых в проверяемой спецификации влияет на другие системы (волновой эффект).
- Постарайтесь ограничить команду шестью членами или меньше. Это означает, что некоторые точки зрения не будут представлены при каждой проверке. Большие команды могут легко увязнуть в посторонних дискуссиях, попытках решения проблем и дебатов о том, что считать ошибкой. При этом снижается темп проверки и растет стоимость обнаружения каждого дефекта.

Роли экспертов

Все участники экспертизы, включая автора, ищут недостатки и возможности улучшения. Отдельные участники при этом играют различные роли.

Автор. Автор создает или поддерживает проверяемый продукт. В роли автора спецификации требований к ПО, как правило, выступает аналитик, который осуществлял сбор требований клиента и писал спецификацию. В ходе неофициальных проверок, таких, как критический анализ, именно автор часто возглавляет дискуссию. Однако в экспертизе он принимает более пассивное участие. Он может не брать на себя обязанности других участников — координатора, читателя или секретаря. Поскольку автор не принимает активное участие и, следовательно, оставляет самолюбие за дверью, он имеет возможность выслушать других участников инспектирования, ответить — но не вступать в дебаты — на их вопросы и поразмыслить. Зачастую автору удается обнаружить ошибки, не замеченные другими участниками.

Координатор. Координатор, или ***руководитель проверки,*** планирует экспертизу совместно с автором, согласовывает особенности и организует совещания. Координатор распределяет проверяемый материал между участниками за несколько дней до совещания. Он отвечает за своевременное начало совещаний, поощрение вклада каждого участника и управление дискуссией, которая должна быть направлена на выявление дефектов, а не на решение проблем. Кроме того, в его обязанности входит информировать менеджеров или того, кто занимается сбором результатов различных экспертиз, о результатах. Координатор также работает вместе с автором над предложенными изменениями, чтобы все дефекты и проблемы, поднятые в ходе инспектирования, были рассмотрены должным образом.

Читатель. Один из проверяющих выступает в роли читателя. На совещании он перефразирует положения спецификации — по одному требованию за раз. Затем другие участники указывают на потенциальные дефекты и проблемы. Выражая требование своими словами, читатель дает трактовку требования, которая может отличаться от интерпретации других членов команды. Это хороший способ выявления неясностей, возможных недостатков или предположений. При этом важно, чтобы роль читателя выполнял не автор.

Секретарь. Секретарь, или ***клерк,*** использует стандартные формы для документирования возникающих вопросов и дефектов, выявленных в ходе совещания. Секретарь должен вслух

прочитать свои записи, чтобы убедиться в их точности. Другие участники инспектирования должны помочь ему зафиксировать суть каждой проблемы таким образом, чтобы автор смог четко уяснить природу и местоположение каждой проблемы.

Входные критерии

Вы готовы к экспертизе документа о требованиях в том случае, если он удовлетворяет определенным предварительным условиям. Набор входных ***критериев*** (entry criteria) четко обрисовывает авторам то, как следует подготовиться к инспектированию. Они также удерживают команду от траты времени на проблемы, которые следует решать до начала экспертизы. Координатор использует входные критерии в качестве контрольного списка до того, как принять решение о начале инспектирования. Вот несколько входных критериев для составления документации по требованиям:

- документ должен соответствовать шаблону;
- в документе выполнена проверка правописания;
- автор просмотрел документ на наличие ошибок в макете;
- для экспертизы подготовлены все необходимые проверяющим лицам предшествующие документы или документы, на которые имеются ссылки;
- номера строк следует напечатать в документе, чтобы облегчить ссылки на определенные элементы в течении совещания;
- все нерешенные вопросы помечаются значком «**TBD**» (to be determined — необходимо определить);
- координатор выявляет не более трех существенных дефектов после 10-минутной проверки выбранного фрагмента документа.

Этапы экспертизы

Экспертиза — это многоэтапный процесс, как показано на рис. 15-2. Цель каждого этапа кратко описана далее в этом разделе.

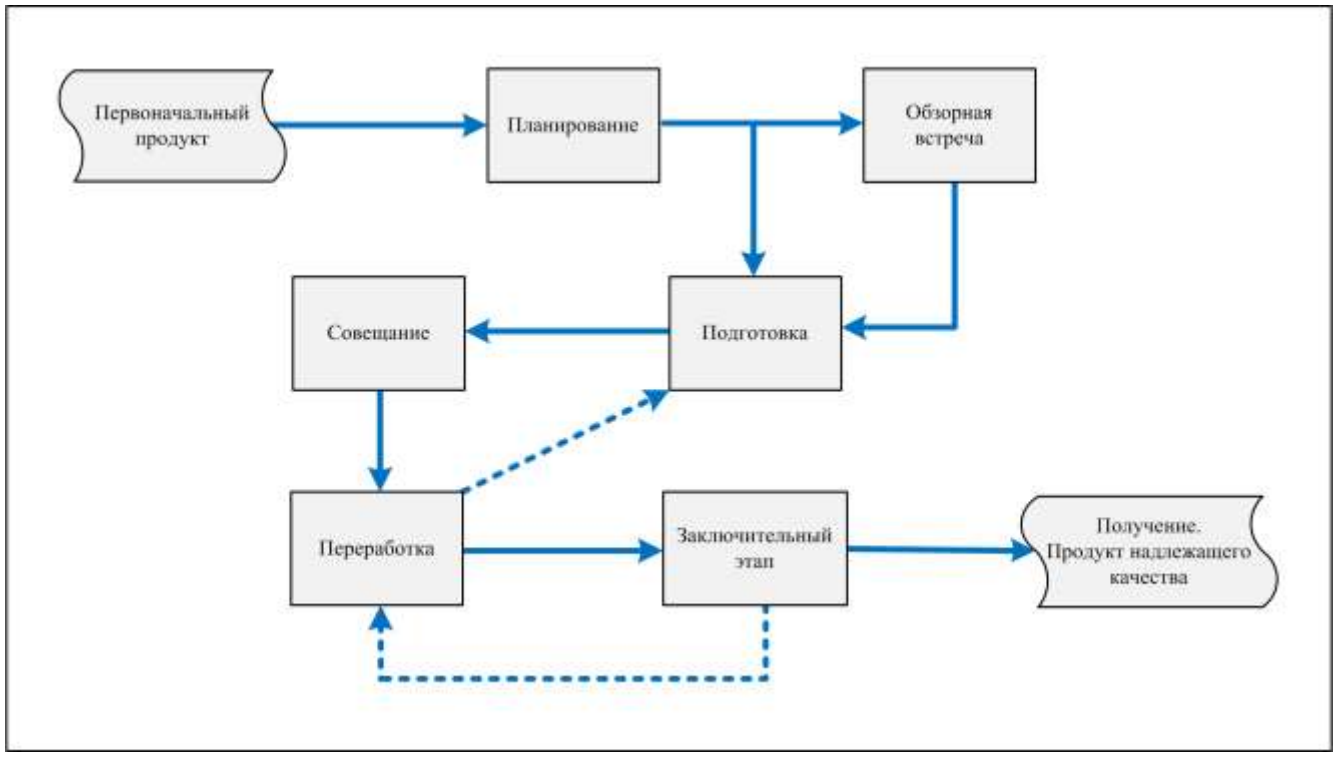


Рис. 15-2. Экспертиза — это многоэтапный процесс. Пунктирные линии указывают на то, что определенные этапы экспертизы могут быть выполнены несколько раз

Планирование. Экспертизу совместно планируют автор и координатор. Они определяют состав участников, материалы, которые проверяющие должны получить до начала первого

совещания, и количество совещаний, необходимых для охвата всего материала. Количество обнаруженных дефектов очень зависит от темпов работы (Gilb и Graham, 1993). Как видно на рис. 15-3, чем медленнее изучается спецификация требований к ПО, тем больше дефектов удается выявить (весьма распространено альтернативное толкование этой связки: «Темпы проверки снижаются, если вы обнаруживаете много ошибок»). Поскольку ни одна команда не может тратить бесконечное количество времени на проверку требований, выберите подходящий темп, принимая во внимание риск пропуска важных дефектов. Чаще всего — от двух до четырех страниц в час, хотя оптимальная скорость, при которой достигается максимальный эффект ниже примерно в два раза (Gilb и Graham, 1993). При выборе темпа работы учитывайте следующие факторы:

- дату предыдущей экспертизы;
- объем текста на каждой странице;
- сложность спецификации;
- риск не заметить ошибку;
- насколько критично для успеха проекта проверить весь этот материал;
- квалификацию автора спецификации требований к ПО.

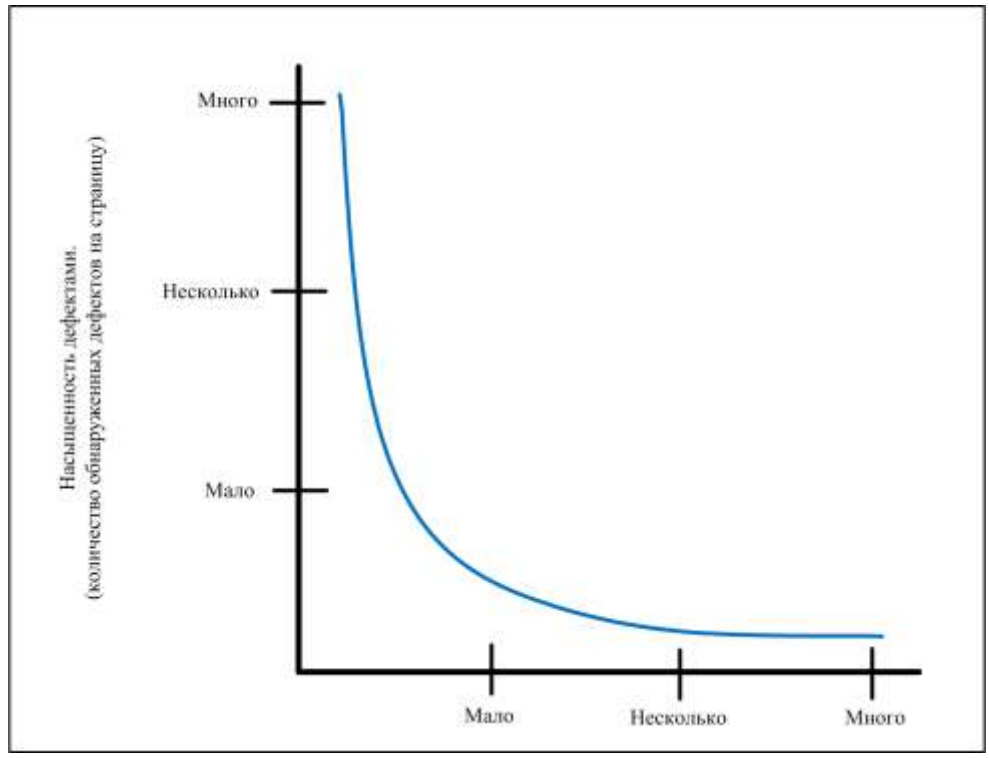


Рис. 15-3. Количество обнаруженных дефектов зависит от темпа инспектирования

Обзорная встреча. Во время обзорной встречи автор описывает исходные данные материала, предназначенного для проверки, предположения, которые он сделал, и его личные цели. Если все проверяющие хорошо знакомы с проверяемым материалом, вы можете пропустить эту стадию.

Подготовка. До совещания каждый из экспертов исследует продукт, чтобы определить возможные дефекты и возникающие вопросы; для этого они используют контрольные списки типичных дефектов (они описаны далее в этой главе) и другие приемы анализа (Wiegers. 2002a). До 75% дефектов, обнаруживаемых при экспертизе, выявляются на этапе подготовки (Humphrey, 1989), поэтому этот этап пропускать нельзя.

Дополнительная информация

Приемы по выявлению недостающих требований, описанные в главе 7, могут оказаться полезными при подготовке к экспертизе.

Ловушка

Не следует начинать совещание, если участники еще не изучили продукт самостоятельно. Неэффективные совещания могут породить мнение, что экспертиза не что иное, как пустая трата времени.

Написанные документы требований никогда не смогут заменить дискуссий между участниками проекта, однако серьезные недостатки спецификации необходимо исправить. Возможно, стоит обратиться к разработчикам, проверяющим спецификацию требований к ПО, с просьбой оценить ясность (понятность) требований по 5-балльной шкале (Pfleeger, 2001). Оценка «1» означает, что разработчик не имеет ни малейшего представления о содержании требования, а «5» — что оно абсолютно понятное, полное и недвусмысленное.

Совещание. На совещании читатель знакомит членов команды с каждым требованием спецификации — по одному за раз, перефразируя их своими словами. По мере обсуждения дефектов и других проблем секретарь заносит их в форму, которая для автора требований становится руководством к действию. Цель совещания — выявить в документе как можно больше ошибок. Совещание не должно длиться более двух часов, поскольку работу уставших людей нельзя назвать продуктивной. Если же вам нужно больше времени, следует запланировать еще одно совещание.

В заключение совещания команда принимает решение: следует ли принять документ без изменений, с незначительными поправками или указать на необходимость значительной переработки. Решение ***«необходимы значительные изменения»*** свидетельствует о наличии серьезных проблем в процессе разработки требований. Возможно, стоит изучить уже проделанную работу и попытаться найти выход до того, как приступать к следующему этапу работы над спецификацией (Kerth, 2001).

Иногда проверяющие сообщают только о поверхностных и незначительных проблемах. Вдобавок проверяющие легко отвлекаются на дискуссии о том, действительно ли обсуждаемый вопрос является дефектом, на споры о границах проекта и на обсуждение возможных решений проблем. Это все, конечно, полезно, однако эти действия отвлекают проверяющих от непосредственной цели совещания — выявления серьезных дефектов и возможностей улучшения.

Некоторые исследователи придерживаются мнения, что совещания слишком трудоемки (Votta, 1993). Я же считаю, что на них можно выявить те дефекты, которые не удастся обнаружить проверяющим при индивидуальной работе. Как и в том, что касается качества, вам придется принимать рискованные решения о том, сколько усилий вы готовы затратить на улучшение требований перед началом разработки и сборки продукта.

Переработка. Практически на каждом этапе работы, связанным с контролем качеством, выявляются определенные дефекты. После совещания автору следует отвести определенное время на переработку требований. Исправление в будущем дефектов требований обойдется дороже, поэтому именно сейчас следует разрешить неясности, устранить двусмысленности и заложить основу успешного проекта по разработке. Не имеет смысла проводить проверку, если вы не собираетесь исправлять ошибки, которые будут выявлены.

Заключительный этап. На этом заключительном этапе экспертизы координатор или специально назначенный член команды работает с автором, чтобы убедиться, что все поднятые проблемы разрешены и все ошибки исправлены соответствующим образом. Заключительный этап завершает процесс инспектирования, на этом этапе координатор определяет, удовлетворяют ли выходные критерии экспертизы.

Выходные критерии

В ходе экспертизы определяются *выходные критерии* (exit criteria), которые необходимо одобрить, прежде чем координатор объявит о том, что проверка закончена. Вот перечень стандартных критериев:

- все возникшие вопросы были сняты;
- все изменения, внесенные в документ и соответствующие продукты, были внесены

корректно;

- в измененном документе проверено правописание;
- все проблемы, помеченные «TBD», разрешены, при этом фиксировались данные по разрешению каждого, дата и исполнитель;
- документ был сверен с помощью системы управления конфигурацией проекта.

Контрольные списки дефектов

Для того чтобы помочь экспертам обнаружить типичные ошибки в инспектируемых продуктах, разработайте контрольный список дефектов для каждого типа документа о требованиях, создаваемого в вашей компании. С помощью таких контрольных списков вы привлекаете внимание проверяющих к часто возникающим проблемам с требованиями. На рис. 15-4 показан контрольный список для проверки варианта использования, а на рис. 15-5 контрольный список для проверки спецификации требований к ПО.

Весь длинный контрольный список запомнить невозможно. Сократите списки, сообразуясь с нуждами вашей компании, и измените их в соответствии с тем, какие проблемы в ваших требованиях повторяются чаще всего. Вы можете попросить некоторых инспекторов при подготовке использовать разные поднаборы контрольных списков. Одного проверить, верны ли перекрестные ссылки внутренней документации, другого — могут ли требования служить основой для дизайна, а третьего — оценить проверяемость требований. Проведенные исследования показали, что такое распределение обязанностей— естественно, когда экспертов снабжают четкими указаниями или сценариями, которые помогают им при поиске ошибок— более эффективно, чем когда эксперты работают по одинаковым контрольным спискам. (Porter, Votta и Basili, 1995).

1	Является ли конкретный вариант использования продукта автономной и отдельной задачей?
2	Ясно ли сформулирована цель или измеряемое значение для конкретного варианта использования продукта?
3	Очевидно ли, кто получит преимущества от этого варианта использования?
4	Написан ли вариант использования на базовом уровне, без ненужных деталей, связанных с дизайном и реализацией?
5	Все ли предполагаемые альтернативные направления задокументированы?
6	Все ли известные условия исключения задокументированы?
7	Есть ли какие-либо последовательности действий, которые можно разделить на отдельные варианты использования?
8	Все ли этапы каждого направления записаны в ясной, недвусмысленной и полной форме?
9	Каждый ли исполнитель и стадия варианта использования продукта соответствуют выполнению конфетной задачи?
10	Осуществимо ли и поддается ли проверке каждое альтернативное направление, определенное в варианте использования?
11	Должным ли образом структурируют вариант использования выходные и выходные условия?

Рис. 15-4. Контрольные списки дефектов для документов о вариантах использования

	Организация и завершенность
1	Корректны ли все ли внутренние перекрестные ссылки на другие документы?
2	Написаны ли все ли требования на одном и том же соответствующем уровне детализации?
3	Обеспечивают ли требования адекватную основу для дизайна?
4	Указан ли приоритет реализации каждого требования?

5	Определены ли все внешние интерфейсы оборудования, ПО и взаимодействия?
6	Определены ли все алгоритмы, соответствующие функциональным требованиям?
7	Включены ли в спецификацию требований к ПО все известные потребности клиента или системы?
8	Отсутствует ли в требовании какая-либо необходимая информация? Если да, помечена ли она значком «TBD»?
9	Задокументировано ли ожидаемое поведение системы для всех предполагаемых ошибочных условий?
	Корректность
1	Конфликтуют ли какие-либо требования с другими требованиями или повторяют их?
2	Написано ли каждое требования ясным, точным и недвусмысленным языком?
3	Можно ли проверить каждое требование с помощью тестирования, демонстрации, просмотра или анализа?
4	Не выходит ли какое-либо требование за границы проекта?
5	Нет ли в каком-либо требовании тематических или грамматических ошибок?
6	Можно ли реализовать все требования, не выходя за рамки установленных ограничений?
7	Все ли перечисленные сообщения об ошибках уникальны и выразительны?
	Атрибуты качества
1	Все ли задачи, связанные с производительностью, сформулированы соответствующим образом?
2	Все ли соображения, касающиеся охраны труда и безопасности, сформулированы соответствующим образом?
3	Приведены ли все ли соответствующие задачи, связанные с атрибутами качества, ясно задокументированы и подсчитаны, с учетом приемлемых компромиссов?
	Возможность отслеживания
1	Каждое ли требование идентифицировано уникально и корректно?
2	Прослежено ли каждое функциональное требование до более высокого уровня (например, требования к системе или вариант использования)?
	Особые проблемы
1	Все ли требования можно отнести к именно к требованиям, а не к решениям разработки или реализации?
2	Идентифицированы ли все функции, зависящие от времени, и определены ли их критерии?
3	Были ли рассмотрены соответствующим образом все проблемы, связанные с интернационализацией?

Рис. 15-5. Контрольные списки дефектов для спецификации требований к ПО

Проблемы при просмотре требований

Экспертные оценки можно отнести как к техническим приемам, так и к работе с людьми. Просьба коллег высказать их мнение о том, что не так с вашей работой, — сознательное, а не интуитивное действие. Для введения практики экспертных оценок в компанию, специализирующуюся на разработке ПО, необходимо время. Ниже описаны типичные проблемы, с которыми специалисты компании столкнутся при проверке документации требований, и предложения по их решению (Wiegers, 1998a; Wiegers, 2002a).

Большой объем документации.

Перспективу тщательной проверки спецификации требований к ПО, состоящей из нескольких сотен страниц, можно смело назвать ужасающей. Вам покажется заманчивым пропустить проверку и сразу приступить к конструированию — тоже не самый мудрый выбор. Даже в спецификации среднего размера, как правило, все эксперты тщательно проверят первую часть, несколько наиболее стойких изучат середину, но маловероятно, чтобы кто-то дошел до конца.

Чтобы избежать перегрузок, проверяйте документ по мере его разработки, не ждите, когда он будет закончен. Определите области повышенного риска, которые требуют тщательной экспертизы — для менее важных фрагментов хватит и неофициального просмотра. Попросите определенных инспекторов начать с различных частей документа, это позволит окинуть каждую страницу свежим взглядом. Возможно, следует использовать несколько небольших команд для проверки различных фрагментов документа, однако при этом вы рискуете не заметить несогласованности. Чтобы оценить, требуется ли проводить экспертизу всей спецификации, исследуйте репрезентативную выборку фрагментов. Изучив количество и типы обнаруженных ошибок, вы определите, стоит ли проводить полную проверку (Gilb и Graham, 1993).

Большая команда экспертов. Многие участники проекта и клиенты хотят заниматься требованиями, поэтому список желающих может оказаться весьма длинным. Однако чем больше команда экспертов, тем труднее составить график совещаний, тем больше на них посторонних разговоров и тем труднее прийти к согласию по каждому вопросу.

Однажды я вместе с 13 инспекторами был приглашен на совещание, организованное человеком, который явно не понимал важность небольшой команды. Четырнадцати персонам трудно договориться, как выйти из горящей комнаты, не говоря уже о принятии решения, правильно ли данное требование. Попробуйте придерживаться следующих советов при работе с потенциально большими командами инспекторов:

- убедитесь, что каждый участник понимает, что его задача — выявлять дефекты, а не повышать квалификацию или отстаивать позицию;
- четко представьте себе, чью точку зрения (клиента, разработчика или тестировщика) представляет каждый участник. Тактично отклоните участие лиц, готовых отстаивать точку зрения, которая уже представлена. Если несколько человек относятся к одной группе, они могут объединить свою информацию и делегировать для участия в совещании кого-то одного;
- соберите несколько небольших групп для параллельной проверки спецификации требований к ПО и объедините их контрольные списки дефектов в один, удалив все повторы. Исследования показали, что экспертиза документа о требованиях, выполняемая несколькими командами, выявляет больше ошибок, чем одна большая команда (Martin и Tsai, 1990; Schneider, Martin и Tsai, 1992; Kosman, 1997), Как правило, результаты параллельных проверок скорее дополняют, чем повторяют друг друга.

Географическое разделение инспекторов. Все больше компаний разрабатывают продукты, привлекая команды, разъединенные географически. В этом случае проверки проводить труднее. Видеоконференции могут стать эффективным решением, но в телеконференциях вы не можете воспринимать язык тела и мимику других участников, как при личных встречах.

Просмотр электронного файла, размещенного в общей сетевой папке, — метод, альтернативный традиционным совещаниям экспертов (Wiegers, 2002a). В этом случае рецензенты используют функции текстового редактора, чтобы ввести свои комментарии в проверяемый документ. Каждый комментарий помечается инициалами эксперта, таким образом, любой может ознакомиться с мнением других рецензентов. ПО, поддерживающее совместную работу через Web встроено в такие инструменты как, ReviewPro компании Software Development Technologies *(http://www.sdtcorp.com).* Если вы решите не проводить совещания, то отдавайте себе отчет, что при этом результативность инспектирования снизится примерно на 25% (Humphrey, 1989), однако стоимость экспертизы также уменьшится.

Тестирование требований

Трудно представить себе, как система будет функционировать при определенных условиях только на основании прочитанной спецификации требований к ПО. Варианты тестирования, созданные на основании функциональных или пользовательских требований, помогают участникам проекта получить представление об ожидаемом поведении системы. Просто разработка вариантов тестирования поможет выявить множество проблем с требованиями, даже если вы не станете выполнять тесты (Beizer 1990). Если вы начнете разрабатывать варианты тестирования

сразу после стабилизации некой части требований, вам удастся обнаружить проблемы, которые еще можно устранить с минимальными затратами.

Ловушка

Остерегайтесь тестировщиков, которые утверждают, что не могут приступить к работе, пока требования еще формируются, и тех, которые утверждают, что для тестирования ПО им не нужны требования. Тестирование и требования связаны между собой синергетическими отношениями, поскольку они представляют дополняющие взгляды на систему.

При написании вариантов тестирования (функциональность) кристаллизуется ваше понимание того, как система должна себя вести при определенных условиях. Неясные и двусмысленные требования будут отскакивать от вас, поскольку вы не сможете описать ожидаемую реакцию систему. Когда аналитики, разработчики и клиенты вместе создают варианты тестирования, они вырабатывают общее понимание того, как продукт будет работать.

Как сделать Чарли счастливым

Как-то раз я попросил Чарли, профи в написании сценариев для UNIX в моей группе, сконструировать простое расширение интерфейса электронной почты для коммерческой системы отслеживания дефектов, которой мы пользовались. Я написал дюжину функциональных требований, описывающих, как должен работать интерфейс электронной почты. Чарли пришел в возбуждение. Он создал множество сценариев, но до сих пор не видел ни одного требования в письменной форме. К сожалению, я промедлил пару недель, прежде чем написал варианты тестирования для этой функции электронной почты. Разумеется, я допустил ошибку в одном из требований. Я обнаружил ее, поскольку мое представление о том, как функция должна работать, раскрытое в 20 контрольных примерах, оказалось несовместимо с одним из требований. Раздосадованный, я исправил требование с ошибкой, еще до того, как Чарли завершил разработку, и в законченном варианте не было ни одной ошибки. Это маленькая победа, но даже маленькие победы имеют смысл.

Вы можете заняться созданием концептуальных вариантов тестирования, основываясь на вариантах использования продукта или других представлениях пользовательских требований, уже на ранней стадии процесса разработки (Ambler, 1995; Collard, 1999; Armour и Miller, 200"). Также варианты использования пригодятся вам для оценки текстовых требований, моделей анализа и прототипов. Варианты тестирования должны охватывать нормальную линию поведения варианта использования продукта, альтернативные направления, а также исключения, идентифицированные в ходе сбора информации и анализа. Эти концептуальные (или абстрактные) варианты тестирования не зависят от реализации. Для примера рассмотрим вариант использования «Просмотр заказа» (View Order) для Chemical Tracking System. Концептуальные варианты тестирования могут быть следующими:

- пользователь вводит номер заказа, который он хочет просмотреть - этот заказ существует - пользователь разместил заказ. **Ожидаемый результат:** ***отображение заказа детально***;
- пользователь вводит номер заказа, который он хочет просмотреть - этот заказ не

существует. **Ожидаемый результат:** ***появляется сообщение: «К сожалению, заказ не найден»;***

- пользователь вводит номер заказа, который он хочет просмотреть - этот заказ существует - но он размещен не этим пользователем. **Ожидаемый результат:** ***появляется сообщение: «К сожалению, это не ваш заказ».***

В идеале разработчик пишет функциональные требования, а тестировщик — варианты тестирования на основе одних и тех же материалов (одна исходная точка на рис. 15-6) — пользовательских требований. Неясности в пользовательских требованиях и различные интерпретации выливаются в несогласованность функциональных требований, моделей и вариантов тестирования. По мере того, как разработчики преобразовывают требования в пользовательский интерфейс и технический дизайн, тестировщики могут переработать концептуальные тесты в детально проработанные процедуры тестирования для официального тестирования системы (Hsia, Кипди Sell, 1997).

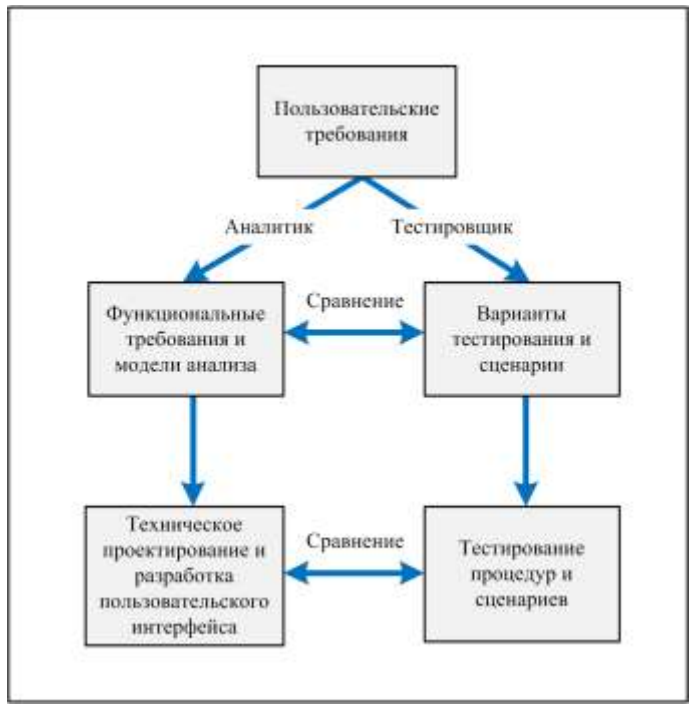


Рис. 15-6. Разработка и тестирование продуктов имеют один общий источник

Сначала вам может показаться, что тестирование требований — абстрактная категория. Давайте посмотрим, как команда разработчиков Chemical Tracking System связала вместе спецификации требований, моделирование анализа и первую стадию создания вариантов тестирования. Далее рассматриваются бизнес-требование, вариант использования продукта, некоторые функциональные требования, часть карты диалогов, а также вариант тестирования, которые связаны с задачей запроса химиката.

Бизнес-требование. Одна из основных бизнес-целей Chemical Tracking System такова:

«Эта система сэкономит компании в первый год применения 25% затрат на химикаты, позволив полностью использовать химикаты, уже имеющиеся в распоряжении компании…»

Дополнительная информация

Бизнес-требование взято из документа об образе и границах проекта, описанного в главе 5.

Вариант использования продукта. Вариант использования продукта, поддерживающий это бизнес-требование, называется ***«Запрос химиката»*** (Request a Chemical). Он позволяет пользователю запросить, контейнер с химикатом, который уже имеется в наличии на складе химикатов. Вот как звучит описание варианта использования:

«Сотрудник, размещающий заказ на химикат, указывает в запросе желаемый химикат, вводя его название или номер идентификатора химиката или импортируя его структуру из инструмента для рисования. Система удовлетворяет запрос, предлагая новый или бывший в употреблении контейнер с химикатом со склада химикатов, либо позволяя создать запрос для заказа химиката у стороннего продавца»

Дополнительная информация

Этот вариант использования продукта показан на рис. 8-6.

Функциональное требование. Здесь показана некоторая функциональность, связанная с этим случаем использования продукта.

1. Если на складе есть контейнеры с химикатами, на которые поступил запрос, система отобразит список доступных контейнеров;
2. Пользователь либо выберет один из перечисленных контейнеров или попросит разместить запрос на заказ нового контейнера у продавца.

Карта диалогов. На рис. 15-7 показана часть карты диалогов для варианта использования «Запрос химиката», которые имеют отношение к этой функции. Прямоугольники на этой карте диалогов представляют изображения пользовательского интерфейса, а стрелки показывают возможные пути перехода от одного изображения к другому.

Дополнительная информация

Подробнее о картах диалогов рассказано в главе 11.

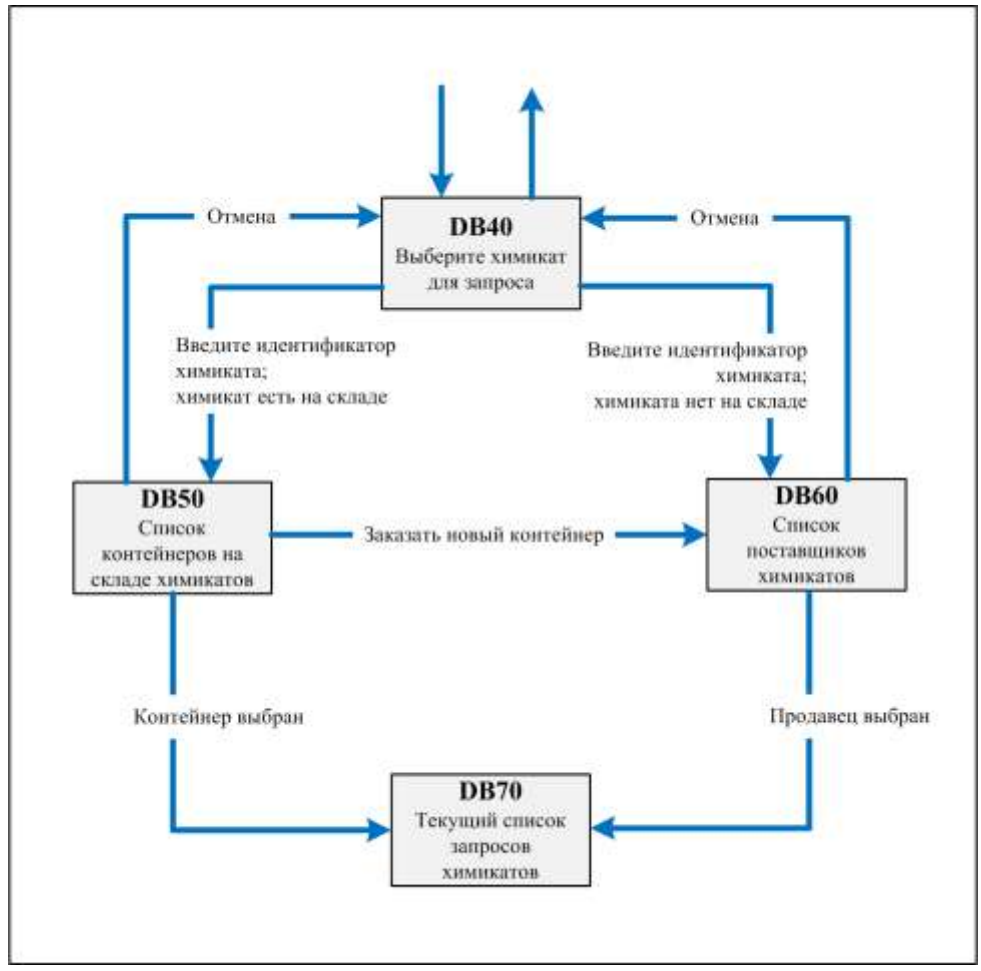


Рис. 15-7. Фрагмент карты диалогов для варианта использования «Запрос химиката»

Контрольный пример. Поскольку для этого варианта использования существует несколько возможных путей выполнения, вы можете придумать несколько вариантов тестирования для обработки нормального направления, альтернативных направлений и исключений. Ниже показан вариант тестирования, когда пользователю отображаются контейнеры, доступные на складе. Этот вариант тестирования разработан на основе описания варианта использования этой задачи пользователя и карты диалогов с рис. 15-7.

«В диалоговом окне DB40 введите действительный идентификатор химиката; на складе имеются два контейнера с этим химикатом. Откроется диалоговое окно IDB50, в котором показаны два контейнера. Выберите второй контейнер. DB50 закроется, и контейнер 2 будет добавлен в конец списка текущих запросов на химикаты в диалоговом окне DB70»

Рамеш, руководитель тестирования Chemical Tracking System, написал несколько вариантов тестирования, похожих на этот, исходя из своего понимания того, как пользователь может взаимодействовать с системой для запроса химиката. Такие абстрактные варианты тестирования не зависят от деталей реализации. В них не разъясняется ввод данных в поля, щелчки кнопок или другие особые детали взаимодействия. По мере разработки пользовательского интерфейса Рамеш возвращался к этим абстрактным вариантам тестирования, создавая более точные процедуры тестирования.

А теперь самое интересное — тестирование требований с помощью вариантов тестирования. Сначала Рамеш соединил на карте каждый вариант тестирования с функциональными требованиями. Он проверил и убедился, что вариант тестирования может быть выполнен при существующем наборе требований. Он также убедился, что по крайней мере один вариант тестирования связан с каждым функциональным требованием. Затем Рамеш проследил путь исполнения каждого варианта тестирования по карте диалога с помощью маркера. Толстая линия на рис. 15-8 показывает, путь предыдущего варианта тестирования на карте диалогов.

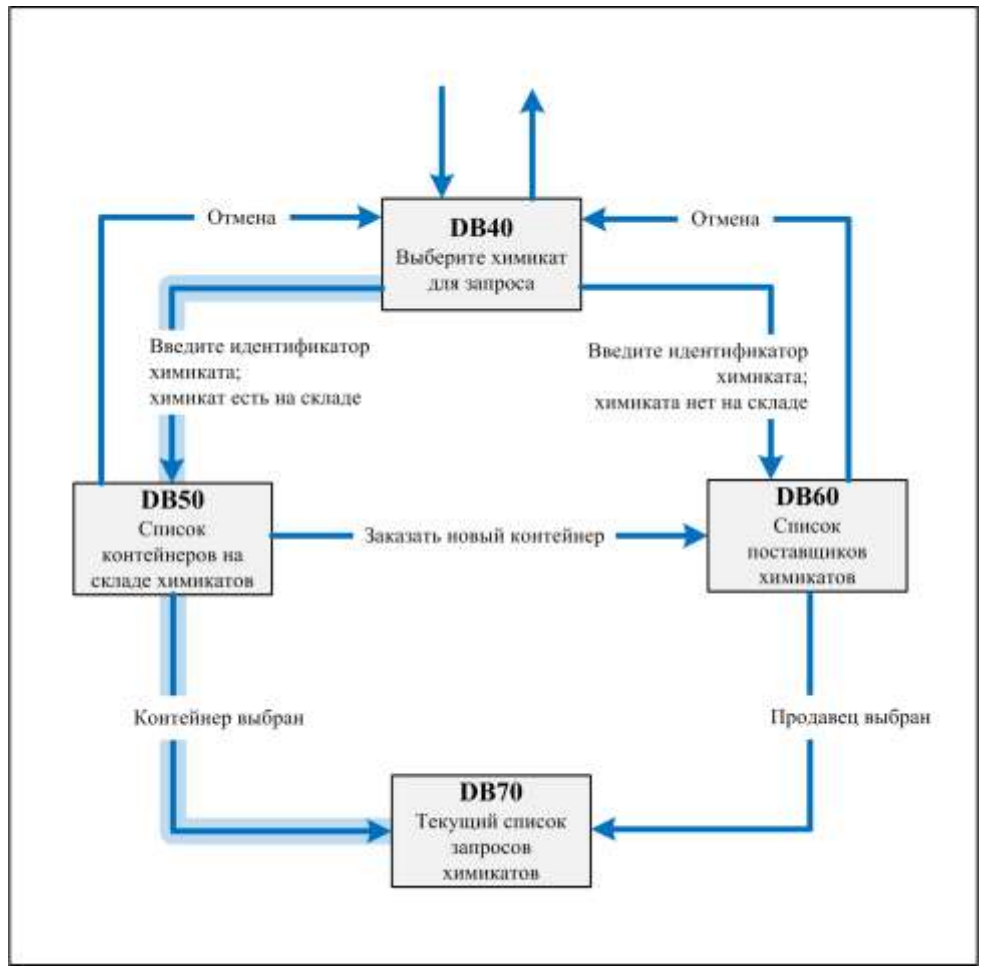


Рис. 15-8. Отслеживание варианта тестирования на карте диалогов для варианта использования «Запрос химиката»

Отслеживая путь исполнения для каждого варианта тестирования, вы можете найти некорректные или пропущенные требования, исправить ошибки на карте диалогов и отшлифовать варианты тестирования. Предположим, что после «исполнения» всех вариантов тестирования в этом случае линия на карте диалогов, помеченная как «заказать новый контейнер» от DB50 к DB60 на рис. 15-7 не была выделена. Возможны два объяснения этому:

- перемещение от DB50 к DB60 не допускается поведением системы. Аналитик должен убрать эту линию с карты диалогов. Если в спецификации есть требование, в котором указан этот переход, то это требование также необходимо удалить;
- перемещение допустимо поведением системы, но вариант тестирования, который демонстрирует это поведение, отсутствует.

Точно так же, предположим, что вариант тестирования, доступный пользователю, позволяет перейти непосредственно от диалогового окна DB40 к DB70. Однако в диалоговом окне на рис. 15-7 такой линии нет, и поэтому вариант тестирования не может быть выполнен с существующими требованиями. И опять же возможны две интерпретации, и вам необходимо определить, какая из них верна:

- перемещение от DB40 к DB70 не допускается поведением системы, следовательно, вариант тестирования неправильный;
- перемещение от DB40 к DB70 допускается поведением системы, но на карте диалогов и, возможно, в спецификации требований к ПО отсутствует требование, позволяющее выполнить вариант тестирования.

В этих примерах аналитик и тестировщик объединили требования, модели анализа и варианты тестирования для выявления пропущенных, ошибочных или ненужных требований

задолго до того, как разработчики приступили к написанию кода. Концептуальное тестирование требований к ПО — мощный прием управления затратами и графиком проекта, так как он позволяет выявлять неясности и ошибки в требованиях в начале игры. Каждый раз, когда я использую этот прием, я нахожу ошибки во всех элементах, которые сравниваю друг с другом. Как сказал Ross Collard (1999):

Варианты использования продукта и варианты тестирования отлично взаимодействуют в двух случаях: если варианты использования написаны полно, аккуратно и ясно, то процесс составления вариантов тестирования становится крайне простым; а если варианты использования не в лучшей форме, то попытка составления вариантов тестирования поможет вам отладить вариант использования продукта.

Определение критерия приемлемости

Разработчики ПО могут быть уверены, что они создают идеальный продукт, однако последнее слово за клиентом. Клиенты тестируют продукт, чтобы определить, удовлетворяет ли система ***критерию приемлемости*** (acceptance criteria) (IEEE, 1990). Если да, то клиент платит за продукт, разработанный согласно контракту. Критерий приемлемости — и, следовательно, проверка приемлемости — является показателем, удовлетворяет ли продукт задокументированным требованиям и годится ли он для использования в предполагаемой операционной среде (Hsia, Kung и Sell, 1997). Делегирование разработки тестов на приемлемость пользователям — эффективная стратегия разработки требований. Чем раньше в процессе разработки пользователи продумают тесты на проверку приемлемости, тем скорее удастся отловить дефекты в требованиях и разрабатываемом ПО.

Проверку приемлемости следует сосредоточить на предполагаемых сценариях использования (Steven, 2002; Jeffries, Anderson и Hendrickson, 2001). Ключевым пользователям при принятии решения о способе оценки приемлемости системы стоит принять во внимание наиболее общие и важные варианты использования, когда они решают, каким образом оценивать приемлемость ПО. Тесты приемлемости фокусируются на нормальной линии поведения вариантов использования продукта, а не на более редких альтернативных направлениях или на том, обрабатывает ли система исключения соответствующим образом. При малейшей возможности старайтесь автоматизировать тестирование приемлемости. Это упростит вам жизнь, когда их придется проводить после внесения изменений и добавления дополнительной функциональности в следующих версиях продукта. Тестирование приемлемости также должны затрагивать нефункциональные требования. Они должны подтверждать, что цели, связанные с производительностью, достигаются на всех платформах, система соответствует стандартам легкости и простоты использования и все соответствующие пользовательские требования были реализованы.

> **Ловушка**
>
> Пользовательское тестирование приемлемости не заменит полного тестирования системы на основании требований, при котором тестируются все нормальные пути и пути исключений, а также большое количество комбинаций данных.

Если критерий приемлемости разрабатывается клиентами, то появляется еще одна возможность утвердить наиболее важные требования. На этапе сбора информации это изменение формулировки вопроса с «Что вам нужно делать с помощью системы?» к «Как вы делаете вывод о том, что система удовлетворяет вашим потребностям». Если клиент не может описать, как он оценит, что конкретное требование удовлетворено системой, значит, требование сформулировано недостаточно ясно. Недостаточно просто написать требования. Вы должны убедиться, что именно эти требования и нужны и что они достаточно хороши, чтобы стать основой для проектирования, сборки, тестирования и управления проектом. Планирование тестов приемлемости, неофициальных просмотров требований, экспертизы спецификации требований к ПО и приемов

тестирования требований поможет вам собрать продукт высокого качества быстрее и дешевле, чем когда бы то ни было.

Что теперь?

- Выберите наугад страницу в спецификации требований к ПО вашего проекта. Попросите представителей различных групп заинтересованных в проекте лиц тщательно проверить ее с помощью контрольного списка дефектов на рис. 15-5 на предмет выявления проблем.
- Если в ходе этой пробной проверки обнаружено столько ошибок, что рецензенты забеспокоились о качестве продукта, убедите представителей пользователей и разработчиков проверить всю спецификацию требований к ПО. Чтобы эта проверка была максимально результативна, обучите команду.
- Определите концептуальные варианты тестирования для варианта использования продукта или для той части спецификации требований к ПО, которая еще не реализована в виде кода. Спросите у представителей пользователей, отражено ли, по их мнению, в вариантах тестирования предполагаемое поведение системы. Убедитесь, что вы определили все функциональные требования, которые будут разрешены конкретным вариантом тестирования, и что в спецификации нет лишних требований.
- Совместно со сторонниками продукта определите критерии, которые они и их коллеги будут использовать для оценки приемлемости системы.

Глава 16 Проблемы при разработке специальных требований

В этой книге описана разработка требований для только создаваемого, нового ПО или системы, т.е. речь идет о ***проекте с чистого листа*** *(green-field project).* Однако многие организации тратят массу сил на обслуживание существующих систем или построение следующих версий уже установленного коммерческого продукта. Другие компании создают новые системы с нуля, вместо того чтобы заняться интеграцией, настройкой и расширением готовых коммерческих (коробочных) продуктов (commercial off-the-shelf, COTS) для удовлетворения своих потребностей. Есть еще и такие, что разработку ПО доверяют сторонним организациям. В этой главе описываются различные приемы работы над требованиями для таких типов проектов, а также для неожиданно возникающих проектов с непостоянными и неопределенными бизнес-потребностями.

Требования к проектам по обслуживанию

Обслуживание (maintenance) означает изменения, вносимые в эксплуатируемое в настоящее время ПО. Иногда на обслуживание, которое часто называют ***продолжающейся разработкой*** (continuing engineering или ongoing development), тратится большая часть ресурсов организации, специализирующейся на разработке ПО. Программисты, занимающиеся поддержкой, обычно исправляют ошибки, добавляют новые функции или сообщения к существующим системам, а также приводят функциональность в соответствие с изменившимися бизнес-правилами. Немногие действующие системы имеют адекватную документацию. Тех разработчиков, которые стояли у истоков системы и держали всю информацию в головах, давно уже нет. Приемы, описанные в этой главе, помогут вам разбираться с требованиями для обслуживания и поддержки действующих проектов, чтобы сделать продукт более качественным и лучше выполнять дальнейшее обслуживание (Wiegers, 2001).

Пропавшая спецификация

В спецификации требований для следующей версии сформировавшейся системы часто пишут так: «Новая система должна делать то же, что и старая, кроме того, что будут добавлены новые функции и исправлены ошибки». Как-то аналитик получила именно такую спецификацию для пятой версии важного продукта. Чтобы точно выяснить, что же текущая версия делает, она заглянула в спецификацию требований к четвертой версии этого ПО. К сожалению, по существу там было сказано вот что: «Версия 4 должна делать то же, что и версия 3, кроме того, что будут добавлены новые функции и исправлены ошибки». Она подняла все предыдущие документы, но так и не смогла найти настоящую спецификацию к требованиям. В каждой спецификации пречислялись отличия новой версии от предыдущей, однако нигде не было описания первоначальной системы. Как следствие, у каждого сложилось свое понимание возможностей текущей системы. Если вы оказались в такой ситуации, задокументируйте требования для следующей версии более подробно, чтобы все заинтересованные лица смогли разобраться в том, что же все-таки система делает.

Начните сбор информации

При отсутствии точной документации к требованиям, специалисты по обслуживанию должны заняться обратной инженерией, т.е. разобраться в функциях системы исходя из ее кода. Я называю эту работу «археологией программного обеспечения». Чтобы она оказалась максимально выгодной, «археологи» должны документировать результаты своих изысканий в форме описания требований или дизайна. Аккуратно фиксируя информацию об определенных частях текущей системы, команда облегчит себе работу по дальнейшему улучшению системы.

Возможно, ваша текущая система представляет собой бесформенный сгусток истории и загадок, как на рис. 16-1. Представьте, что вас попросили реализовать некую новую функциональность в области А на этом рисунке. Начните с документирования новых требований в структурированной, пусть и неполной, спецификации требований к ПО или воспользуйтесь утилитой для управления требованиями. Когда вы добавите новую функциональность, вам придется сообразить, как новые экраны и функции будут взаимодействовать с существующей системой. Эти взаимодействия показаны на рис. 16-1 в виде мостов между областью А и текущей системой. Анализируя ситуацию, вы получите представление о белой области текущей системы — области *B*. Так вы выявите новую информацию, которую необходимо зафиксировать.

Один из полезных приемов — нарисовать карту диалогов для новых экранов, которые вы собираетесь добавить, показав перемещения от существующих элементов экрана и к ним. Вы также можете воспользоваться и другими приемами моделирования: диаграммами классов и взаимодействия, диаграммами потоков данных и диаграммами «сущность-связь». Контекстная диаграмма или диаграмма варианта использования продукта отображает внешние объекты или лица, взаимодействующие с системой. Еще один способ получения недостающий информации — добавлять в словарь данных новые понятия и изменять существующие определения при добавлении новых элементов к системе.

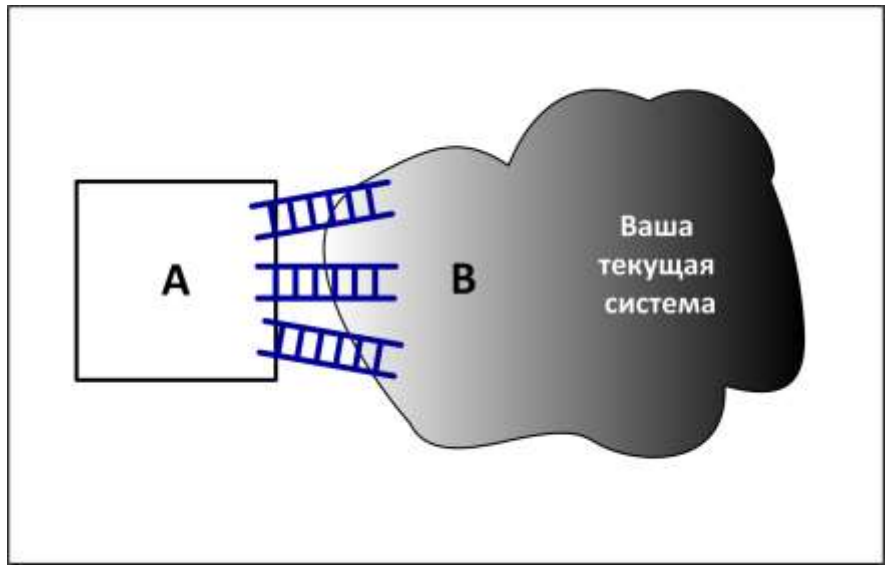


Рис. 16-1. Добавляя ясный и понятный фрагмент А к аморфной документации уже работающей системы, вы проясняете данные в области В документации

При построении представления требований, в которое включены фрагменты текущей системы, решаются три задачи:

- останавливается непроизводительное растрачивание данных, поэтому специалисты по обслуживанию лучше разбираются в изменениях, внесенных только что;
- собирается информация о текущей системе, которая ранее не была задокументирована. При обслуживании системы команда сможет расширять эти раздробленные представления данных, таким образом, непрестанно улучшая документацию к системе. Растущая стоимость документирования этой вновь найденной информацией невелика по сравнению с ценой, которую кому-то придется заплатить в будущем, чтоб «раскопать» ее заново;
- обеспечивается проверка полноты функциональности системы посредством набора действующих тестов. (С другой стороны набор тестов пригодится в качестве первоначального источника информации при восстановлении требований к ПО.)

Если вам нужно оказать хирургическое вмешательство в ПО в связи с вводом новых, пересмотром старых правительственных постановлений или изменением политики компании,

составьте список бизнес-правил, влияющих на систему. Если вы не сделаете этого, вся важная информация останется только в головах членов команды. Поскольку люди по-разному интерпретируют важность влияния бизнеса на ваш проект, раздробленную информацию можно считать жалким вкладом в общее дело.

Множество проблем выявляется на стадии работы над взаимодействующими элементами, включая особенности взаимодействия различных ПО, ПО и оборудования, ПО и людей. При реализации изменений выделите время на документирование обнаруженной информации о текущих внешних пользователях, ПО, оборудовании и коммуникационных интерфейсах. Структурные данные о совместной работе компонентов системы помогут вам в будущем добавлять расширения более безопасно и эффективно. Однажды подрядчику потребовалось внести существенное улучшение в систему управления роботом, над созданием которой работала моя команда. Наши модели дизайна помогли ему, и он смог эффективно совместить новые функции с архитектурой существующей системы. Если бы мы не предоставили эту информацию, подрядчику пришлось заниматься обратной инженерией кода, чтобы разобраться в архитектуре и дизайне.

Разработка полной спецификации требований к ПО для всей производственной системы не всегда целесообразна. Между двумя крайностями — работой безо всякой документацией и воссозданием идеальной спецификации — существует множество промежуточных стадий. Если вы знаете, зачем вам нужны задокументированные требования, то сможете решить, будут ли оправданы затраты на воссоздание всей спецификации или ее части. Чем на более ранней стадии цикла разработки ПО вы находитесь, тем боле ценно наличие задокументированных требований.

Как-то я работал с командой, которая только что начала заниматься требованиями для второй версии крупного продукта со встроенным ПО. Требования для версии, которая в тот момент реализовывалась, были проработаны не очень хорошо. Ведущий аналитик требований желал знать, стоит ли возвращаться и исправлять требования для версии. Руководство же компании полагало, что эта производственная линия будет основных источником дохода в течение по крайней мере 10 лет, а также планировало повторно воспользоваться некоторыми основными требованиями в нескольких побочных продуктах. В этом случае стоило улучшить документацию требований для версии 1, поскольку она лежала в основе всей последующей работы над продуктами этого семейства. Если бы команда в тот момент работала над версией 5.3 предыдущей системы, которую планировалось изъять из производства в течение года, тогда не стоило бы воссоздавать спецификацию требований к ПО.

От кода через требования к тестированию

Компания A. Datum Corporation нуждалась в полном наборе вариантов тестирования для основного продукта, сложного и крайне важного бухгалтерского приложения, разработанного несколько лет назад. Специалисты компании решили воссоздать варианты использования на основе существующего кода, а затем, опираясь на них, разработать варианты тестирования.

Сначала аналитик воссоздал диаграммы классов разработанного ПО с помощью средств, способных создавать модели из исходного кода. Затем он составил описания вариантов использования для типичных пользовательских задач, причем некоторые были весьма сложными. Эти варианты использования затрагивали все возможные пользовательские сценарии, а также множество условий исключений. Аналитики нарисовали диаграммы последовательностей UML (Unified Modeling Language — унифицированный язык моделирования) для того, чтобы связать варианты использования и классы систем (Armour и Miller, 2001),

И последнее: они вручную собрали большой набор вариантов тестирования, чтобы обработать все нормальные и исключительные случаи вариантов использования. Создавая требования и тестируя функциональность посредством обратной инженерии, вы можете быть уверены, что они отражают реальную систему и ее известные модели

Применяйте новые приемы работы с требованиями

В основе всей разработки ПО лежат требования, которые могут быть сформулированы в форме простого запроса на изменение. Проекты по обслуживанию предоставляют возможность испробовать новые методы разработки требований безопасно и в малом масштабе. Заманчиво считать, что небольшое улучшение не означает необходимости написания требований. Давление, которое оказывается при выпуске следующей версии, может заставить вас решить, что у вас нет времени на требования. Но усложнение проектов позволит вам постепенно обучаться. Таким образом, к следующему крупному проекту вы будете чувствовать себя опытным и уверенным, применяя эффективные приемы при разработке требований.

Предположим, отдел маркетинга или клиент требуют добавления к уже сформированному продукту новой функции. Изучите эту функцию с точки зрения вариантов использования, обсудив с инициатором запроса задачи, которые клиенты будут выполнять с ее помощью. Если до этого вы не имели дела с вариантами использования, попробуйте задокументировать их с помощью стандартного шаблона, как показано на рис. 8-6. Если вы опробуете приемы работы с вариантами использования, то снизите первоначальный риск их применения в проекте, разрабатываемом с нуля, поскольку именно от вашей квалификации может зависеть успех или неудача. Вы можете опробовать в малом масштабе и другие приемы при обслуживании продукта, например:

- создание словаря данных (глава 10);
- рисование моделей анализа (глава 11);
- определение атрибутов качества и целей производительности (глава 12);
- построение пользовательского интерфейса и технических прототипов (глава 13);
- проверку спецификации к требованиям (глава 15);
- написание вариантов тестирования на основе требований (глава 15); определение критерия приемлемости для пользователей (глава 15).

Перемещайтесь по цепочке трассируемости

Требования трассируемости данных помогут программисту по обслуживанию определить, какие компоненты следует модифицировать после изменения определенного требования. Однако в

плохо задокументированной действующей системе информации о трассируемости нет. Когда кому-то из вашей команды доведется модифицировать существующую систему, он должен позаботиться о создании частичной матрицы трассируемости требований для того, чтобы связать новые или измененные требования с соответствующими элементами проектирования, кода и вариантами тестирования. Собирать связи трассируемости по мере разработки не сложно, тогда как воссоздать их в завершенной системе — практически нереально.

Дополнительная информация

Глава 20 посвящена трассируемости требований.

Из-за возникновения «кругов на воде» при многих модификациях вам необходимо убедиться, что каждое изменение требования корректно сказывается на продуктах, связанных с разрабатываемым. Отличный способ проверить согласованность изменений — экспертиза. Ниже перечислены четыре точки зрения (им посвящена Глава 15), которые должны представлять эксперты; это касается и тех, кто занимается обслуживанием:

- автора требований - при модификациях или улучшениях;
- клиента, инициирующего запрос, или представителя отдела маркетинга, которые могут подтвердить, что в новых требованиях точно описаны необходимые изменения;
- разработчиков, тестировщиков или других сотрудников, которые будут работать, руководствуясь новыми требованиями;
- людей, на чью работу могут повлиять вносимые изменения.

Обновляйте документацию

Все существующие представления требований следует обновлять по мере обслуживания, чтобы они адекватно отражали возможности модифицируемой системы. Если обновление обременительно, то им часто будут пренебрегать, поскольку занятые люди стараются сразу перейти к следующему запросу на обновление. А устаревшая документация по требованиям и проектированию вряд ли окажется полезной при дальнейшем обслуживании. В индустрии ПО бытует страх того, что написание документации всегда уходит слишком много времени, поэтому вырабатывается условный рефлекс - пренебрегать всяким обновлением документации к требованиям. Однако какова цена того, что вы не обновляете требования и специалисту по обслуживанию (а, может, им станете вы сами!) придется воссоздавать информацию? Ответив на этот вопрос, вы сможете принять толковое решение о том, следует ли исправлять документацию требований при изменении ПО.

Требования для пакетных решений

Даже при покупке готового пакета в качестве части или всего решения для нового проекта вам понадобятся требования. Готовые продукты, как правило, нужно конфигурировать, настраивать, интегрировать и расширять для работы в предполагаемой среде. Для выполнения всех этих действий необходимы требования, которые, кроме того, позволит вам правильно выбрать наиболее подходящий пакет. Один из способов такой оценки предлагает следующее (Lawlis и др., 2001):

- оценить ваши требования по шкале от 1 до 10, чтобы определить важность каждого;
- определить, насколько каждый пакет отвечает вашим потребностям. Используйте оценку 1 для обозначения полного удовлетворения, 0,5 — для частичного удовлетворения и 0 — если вас все не устраивает. Вы можете найти информацию, чтобы выполнить оценку, в литературе о продукте, в ответе поставщика на запрос на предложение (приглашение принять участие в торгах по проекту) или с помощью прямой проверки продукта;

- оцените каждый пакет для нефункциональных требований, важных для ваших пользователей — производительность, легкость и простоту применения, а также другие атрибуты качества, перечисленные в главе 12;
- оцените стоимость продукта, надежность поставщика, поддержку продукта, осуществляемую поставщиком, внешние интерфейсы, которые позволят вам осуществлять расширение и интеграцию продукта, а также совместимость с технологическими требованиями или ограничениями для вашей среды.

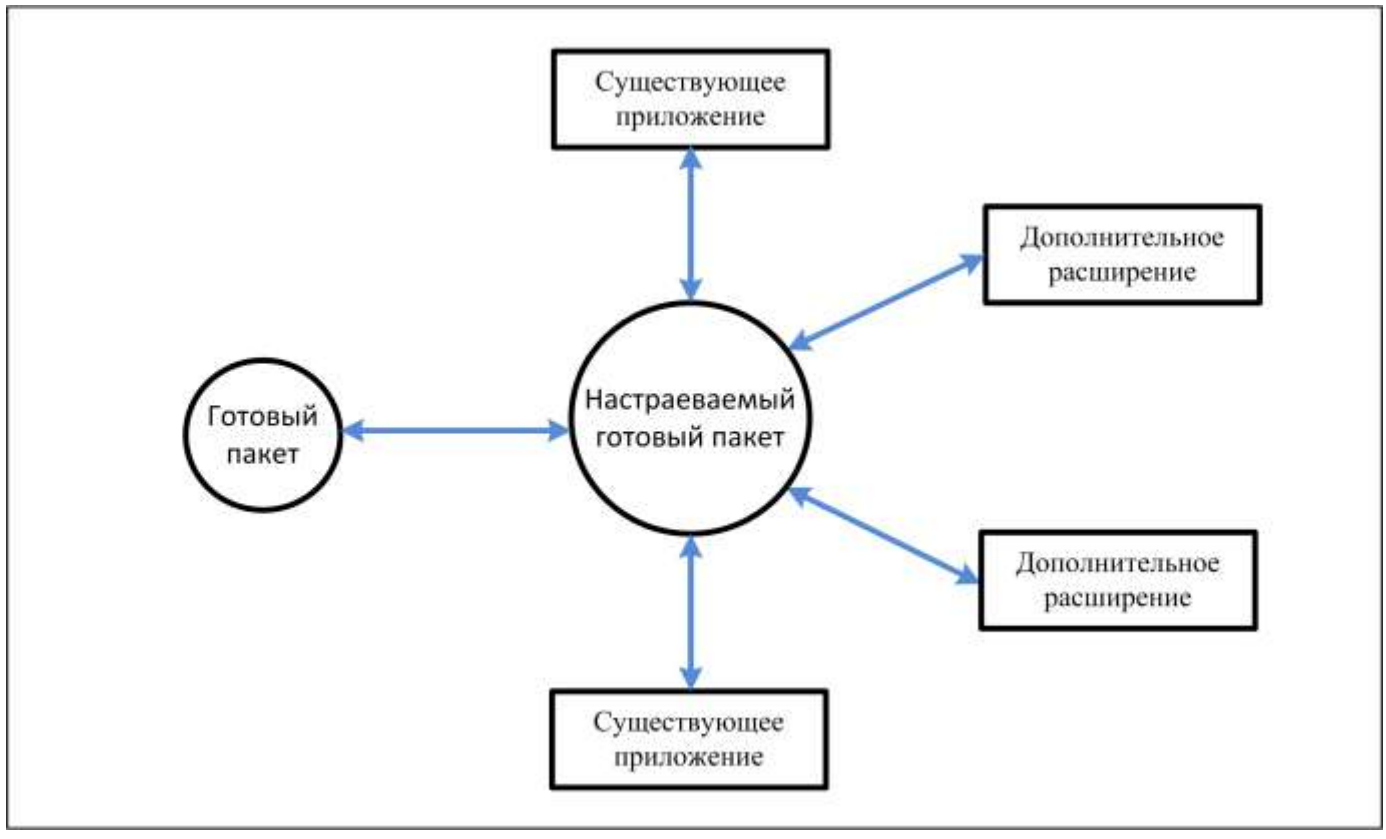


Рис. 16-2. Готовые пакеты настраиваются и интегрируются в существующую среду приложений, и расширяются с помощью дополнительных элементов.

В этом разделе описывается несколько способов определения требований при планировании приобретения коммерческого продукта, отвечающего вашим потребностям (рис. 16-2).

Разработка вариантов использования

Не имеет смысла разрабатывать подробные функциональные требования или пользовательский интерфейс, если вы планируете приобрести готовый продукт. В этом случае стоит обратить особое внимание на уровень требований пользователей. И здесь вам помогут варианты использования продукта. Любой выбранный вами пакет должен обеспечивать выполнение клиентских задач, хотя каждый пакет делает это своим способом. Варианты использования позволят вам обнаружить пробелы и выявить те фрагменты, где необходима настройка или расширение, чтобы пакет полностью удовлетворял вашим потребностям. Многие пакеты предоставляют готовые решения для удовлетворения нужд, связанных с обработкой информации. Следовательно, вы должны указать отчеты, которые должен генерировать готовый продукт, и определить, до какой степени продукт позволит вам настраивать его стандартные отчеты. Вряд ли вы найдете готовые решения для каждого варианта использования, поэтому расставьте их и сообщения по приоритетам. Определите, какие функции должны быть доступны в первый день, какие могут подождать до дальнейших расширений и без каких пользователи не смогут работать.

Для того чтобы определить, позволит ли пакет пользователям выполнять их задачи, и если да,

то насколько хорошо, на основе вариантов использования с высоким приоритетом разработайте вариант тестирования. Включите в них варианты использования, в которых рассматривается, как система обрабатывает возможные важные условия исключений. Схожий прием — запустить готовый продукт для набора сценариев, представляющего ожидаемые модели использования — и еще называют ***оперативным профилем*** *(Musa*. 1996).

Работа с бизнес-правилами

Изучая требования, необходимо определить бизнес-правила, которым готовый продукт должен соответствовать. Сможете ли вы сконфигурировать продукт в соответствии с корпоративной политикой, промышленными стандартами или правительственными постановлениями? Насколько легко вносить коррективы при изменении этих правил и положений? Создаются ли отчеты в корректном формате? Вы также можете определить структуры данных, необходимые для удовлетворения вариантов использования и бизнес-правил. Попробуйте отыскать несогласованности между вашими структурами данных и набором моделей данных поставщика.

Некоторые пакеты учитывают глобальные бизнес-правила, например расчет подоходного налога с дохода или печать деклараций о доходах. Уверены ли вы, что они реализованы корректно? Обеспечит ли, поставщик новой версией пакета в случае изменения этих правил и расчетов? Если да, то как быстро? Предоставит ли поставщик список бизнес-правил, реализованных в пакете. Если продукт поддерживает какие-либо бизнес-правила, которые для вас неприемлемы, можете ли вы отключить или изменить их?

Определение требований к качеству

Атрибуты качества и задачи, связанные с производительностью, о которых рассказывалось в главе 12, — еще один аспект требований пользователей, который имеет большое значение при выборе пакетного решения. Следует рассмотреть, по крайней мере, следующие атрибуты.

Производительность. Каково максимальное время отклика, приемлемое для определенных операций? Может ли пакет обработать ожидаемую загрузку одновременно подключенных пользователей и пропускную способность транзакций?

Легкость и простота использования. Соответствует ли пакет всем принятым стандартам пользовательского интерфейса? Похож ли интерфейс на интерфейсы других приложений, с которыми знакомы пользователи? Насколько легко ваши клиенты смогут обучить работать с пакетом?

Гибкость. Насколько легко разработчики смогут изменить или расширить пакет, чтобы удовлетворить ваши потребности? Входят ли в пакет соответствующие «ловушки» (точки подключения и расширения) и прикладные программные интерфейсы, чтобы добавлять расширения?

Способность к взаимодействию. Насколько легко вы сможете интегрировать пакет с другими приложениями, используемыми в компании? Используются ли стандартные форматы для обмена данными?

Целостность. Допускает ли пакет контроль допуска пользователей к системе или применение специальных функций? Предохраняет ли он данные от потери, повреждения или неавторизованного доступа? Можете ли вы определять различные уровни привилегий пользователей?

При приобретении готовых пакетов организация получает меньшую гибкость при работе с требованиями, чем при разработке продукта «под заказ». Вам необходимо выяснить, какие из запрошенных возможностей не могут быть предметом переговоров, а какие вы можете изменять в рамках ограничений, налагаемых пакетом. Единственная возможность выбрать правильный пакет решений — понять задачи, пользователей и коммерческие задачи, для решения которых вы приобретаете пакет.

Требования к проектам, выполняемым сторонними организациями

Для того чтобы поручить разработку продукта другой компании, необходимо иметь требования высокого качества, поскольку, вероятнее всего, прямые контакты с командой разработчиков будут минимальными. Вы отправите поставщику в виде запроса предложения, спецификацию требований и определенные критерии приемлемости продукта, а они вернут завершенный продукт и сопутствующую документацию, как показано на рис. 16-3.

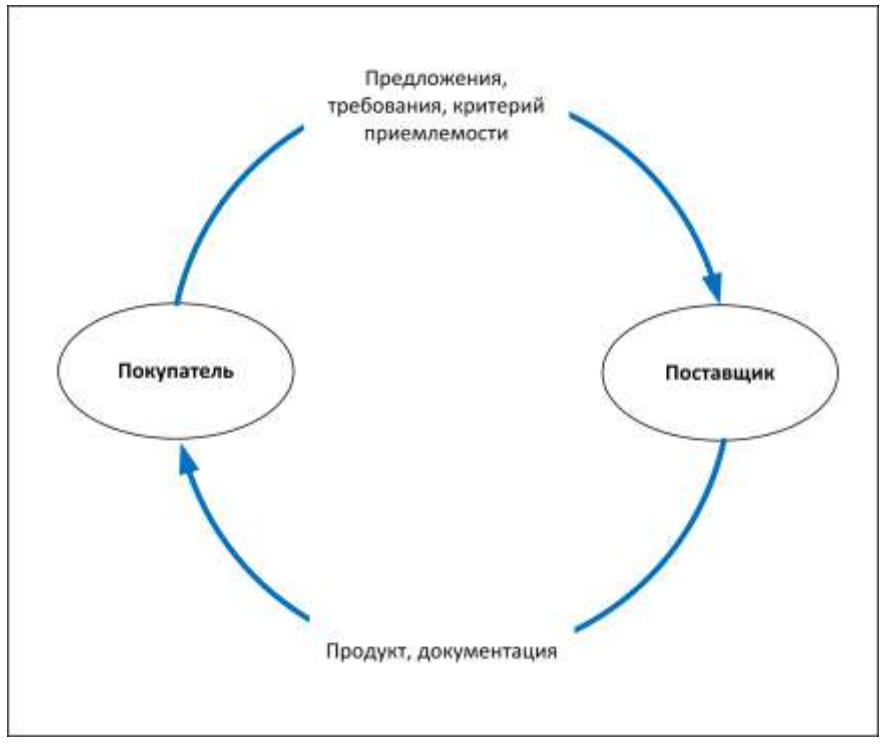


Рис. 16-3. Требования - краеугольный камень проекта, выполняемого сторонними организациями.

У вас не будет возможности для ежедневных уточнений, принятия решений и изменений, которые вы практикуете, когда разработчики и заказчики работают в непосредственной близости друг от друга. Плохо определенные и плохо организованные требования зачастую становятся причиной неудачи проекта, выполняемого сторонними организациями (Wiegers, 2003). При подготовке требований для удаленной разработки вам надо иметь в виду некоторые особенности, которые описаны далее.

Детализируйте запрос. Если вы распространяете запрос на предложение, поставщикам необходимо точно знать, что именно вы хотите, чтобы сообщить вам достоверную информацию и оценку (Porter-Roth, 2002).

Избегайте неясностей. Остерегайтесь неясных терминов (табл. 10-1), они становятся причиной полной неразберихи. Как-то раз я читал спецификацию требований к ПО, предназначенную для сторонней организации-разработчика, в которой много раз повторялось слово «*поддержка*». Однако специалисты этой организации не знали, как значение этого слова интерпретировать в каждом случае. Ответственность за ясное выражение требований заказчика лежит на создателе требований.

Определите точки взаимодействия с поставщиком. Если с разработчиками нельзя организовать встречи «лицом к лицу, в режиме реального времени», придумайте другие механизмы, которые позволят вам быть в курсе дел по вашему проекту. Экспертные оценки и прототипы позволяют понять, как поставщик интерпретирует требования. Поэтапная разработка,

при которой поставщик периодически поставляет небольшие части продукта, считается одним из способов, обеспечивающих управление рисками. Он позволяет быстро корректировать курс, когда разработчики поставщика из-за непонимания выбирают неверное направление.

Ловушка

Не следует предполагать, что поставщики будут интерпретировать неясные и неполные требования точно так же, как вы. Некоторые поставщики понимают требования буквально и создают точно то, что вы запросите. Бремя передачи всей необходимой информации поставщику лежит на заказчике; не отказывайтесь по-чаще беседовать с поставщиком, чтобы снять все вопросы, касающиеся требований.

Определите взаимно приемлемый процесс управления изменениями. В ходе разработки примерно 45% проектов, выполняемых сторонними организациями, существует риск, что пользовательские требования будут неумолимо разрастаться (Jones, 1994). У изменений всегда есть цена, поэтому управление изменениями, позволяющее контролировать границы проекта, жизненно важно при работе по контракту. В контракте необходимо указать, кто платит за различные виды изменений, например за только что запрошенную функциональность или коррективы, вносимые в первоначальные требования.

Выделите время для нескольких циклов и проверки требований.

В график одного неудавшегося проекта, разрабатываемого сторонней организацией, был включен этап длиной в одну неделю под названием «Проведение семинаров по требованиям», следующим был указан этап реализации нескольких подсистем. Поставщик забыл о важнейших промежуточных стадиях — проверке и корректировке спецификации требований. Поскольку процесс разработки требований является итерационным и зависит от эффективного взаимодействия, необходимо предусмотреть время на эти циклы проверок. В том проекте заказчика и разработчика разделял континент. При этом обмен информации при миллиардах вопросов, возникающих при работе над спецификацией требований к ПО, был медленным. Неспособность своевременно решать вопросы, связанные с требованиями, нарушает график проекта, и со временем приведет обе стороны в зал суда.

Определите критерий приемлемости. В соответствии с рекомендацией Stephen Covey «начинайте, думая об окончании» (Covey, 1989), определите, как вы будете оценивать приемлемость разрабатываемого по контракту продукта для вас и ваших клиентов. Как вы будете решать, пора ли делать финальный платеж?

Дополнительная информация

В главе 15 предлагается несколько приемов для определения критерия приемлемости.

Требования для принципиально новых проектов

Когда я слышу выражение «собрать требования», мне представляется поиск требований, лежащих вокруг офиса, как сбор цветов, которые мы можем сорвать и положить в корзину. Если бы это было так просто! В некоторых проектах, где, как правило, проблемы хорошо обоснованы, и в самом деле можно указать большую часть предполагаемой функциональности на ранней стадии. Для них имеет смысл заранее разработать согласованные требования.

Однако в более неопределенных проектах предполагаемые возможности системы проясняются только со временем. Для таких новых проектов характерны неясные требования и частые изменения. Поэтому необходимо и требования, и ПО разрабатывать поэтапно и постепенно (Highsmith, 2000). При быстрой разработке проектов, для которых учитываются растущие требования рынка, часто отходят от общЄ-принятых приемов создания требований. Вместо этого за

основу берут неясные, высокоуровневые положения о бизнес-целях. Однажды президент компании, занимающейся разработкой ПО для Интернета, сообщил мне в разговоре, что у них часто возникают проблемы, связанные с требованиями. Одна из причин крылась в том, что прямо после подписания контракта с клиентом исполнители организовывали семинар, посвященный визуальному дизайну. Они не уделяли времен и на выяснение того, как пользователи смогут получать данные с Web-сайта, что приводило к значительным переделкам.

Преобладание новых и стремительно развивающихся проектов привело к созданию различных методик **быстрой разработки** (*agile development).* Их цель — быстро передать полезную функциональность в руки пользователям (Gilb. 1988; Beck, 2000; Cockburn, 2002). В них применяется упрощенный подход к разработке требований и неформальный подход к управлению требованиями. Требования описываются в терминах характеристик продукта или в форме **пользовательских рассказов (историй)** *(user story),* похожими на обычные варианты использования, описанные в главе 8. Вместо подробной документации к требованиям предпочтение отдается постоянному взаимодействию с представителем клиента, который, находясь рядом, имеет возможность разъяснять детали и отвечать на вопросы.

В философии быстрой разработки изменения ПО рассматриваются как нечто неизбежное и желательное. Система корректируется под воздействием обратной связи с клиентом и при изменении бизнес-нужд, Чтобы справиться с этими изменениям, системы строятся с небольшим приращением, причем подразумевается, что следующая разработанная часть повлияет на уже созданную часть системы, улучшит первоначальные функции и добавит новые. Этот способ хорошо работает при высокой степени неясности в требованиях для информационных систем и Интернет-проектов. Однако он меньше подходит для работы с приложениями, требования к которым ясны, и для разработки встроенных систем.

Совершенно очевидно, что процесс разработки должен быть основан на ясном понимании того, что пользователи хотят делать с помощью системы. Требования к быстро меняющимся проектам слишком нестабильны, чтобы оправдать массу усилий, затраченную на предварительную разработку требований. Однако по мнению Jeff Gainer (1999): «Трудность для разработчиков ПО заключается в том, что при повторяющихся циклах разработки появляется искушение изменить требования и расширить объем работ». Если этими циклами управлять должным образом, то они помогут вам сделать ПО, соответствующее текущим бизнес-нуждам, хотя и ценой переработки заблаговременно завершенного проектирования, кода и тестов.

Иногда лица, заинтересованные в проекте, утверждают, что роль требований неизвестна и непостижима, тогда как они просто горят от нетерпения побыстрее заняться написанием кода, при этом они не учитывают возможность всеобъемлющего перекодирования. Не поддавайтесь искушению использовать «время Интернета» как оправдание экономии на разработке требований. Если вы и в самом деле работаете над совершенно новым проектом, вам, возможно, следует воспользоваться приемами работы с требованиями, описанными в следующих разделах.

Бессистемная спецификация пользовательских требований

Неформально задокументированные требования годятся для систем, создаваемых небольшими, географически не разделенными командами. Методика быстрой разработки, именуемая ***экстремальным программированием*** (Extreme Programming), рекомендует документировать требования в форме простых рассказов пользователей, написанных на учетных карточках (Beck, 2000; Jeffries, Anderson и Hendrickson, 2001). Обратите внимание, что даже при этом подходе необходимо, чтобы клиенты представляли себе требования достаточно ясно, чтобы описать поведение системы в форме рассказов.

Ловушка

Не следует ожидать, что для успеха проекта хватит и телепатической передачи ненаписанных требований. Требования для каждого проекта должны быть представлены в формах, которые позволят ознакомить с ними всех заинтересованных лиц, обновлять их и управлять ими на протяжении разработки проекта. Кроме того, необходимо, чтобы кто-то отвечал за такое документирование и обновление

Присутствие клиента

Частые беседы участников проекта и клиентов считаются наиболее эффективным способом разрешения многих проблем, связанных с требованиями. Написанная документация, несмотря на детализацию, не полностью заменяет это постоянное общение. Основу экстремального программирования составляет постоянное и непосредственное присутствие клиента на этих обсуждениях. Однако, как говорилось в главе 6, большинство проектов предназначено для нескольких классов пользователей, а также для других заинтересованных лиц.

Клиент в поле зрения

Однажды я писал программу для ученого-исследователя, рабочее место которого было расположено примерно в 10 футах от моего стола. Джон мог моментально отвечать на мои вопросы, высказывать свое мнение по поводу экранов интерфейса и пояснять неформально изложенные требования. Однажды Джон перебрался в новый офис, расположенный на том же этаже того же здания. Я заметил, что моя производительность моментально упала из-за задержки при получении ответной информации от Джона. Я стал тратить больше времени на устранение проблем, поскольку иногда двигался в неправильном направлении, так как вовремя не получал корректирующих указаний. При разработке ничто не заменит тесное взаимодействие с работающим за соседним столом клиентом. Однако будьте осторожны — при слишком частых прерываниях людям трудно опять сосредоточиться на работе, Может потребоваться 15 минут на повторное погружение в крайне эффективное, сосредоточенное состояние, которое называется потоком (DeMarcoи Lister, 1999).

Сам по себе находящийся рядом клиент не гарантирует желаемого результата. Мой коллега Крис, налаживая работу команды разработчиков, привлек двух сторонников продукта. Выводы его оказались такими: «В то время как близость оказала благотворное воздействие на команду разработчиков, результаты работы со сторонниками продукта были смешанными. Один из них был в центре всего, и все же умудрялся избегать нас. Человек, выполняющий обязанности сторонника продукта в данный момент, прекрасно общается с разработчиками и действительно убыстряет разработку ПО».

Ловушка

Не следует ожидать, что одному человеку удастся разрешить все возникающие проблемы, связанные с требованиями. Когда роль сторонника продукта (выполняют несколько представителей пользователей, это более эффективно.

Периодическая расстановка приоритетов на ранних стадиях

Поэтапная разработка успешна, если клиенты и разработчики сотрудничают при выборе порядка реализации функций. Задача команды разработчиков — регулярно передавать полезную функциональность и качественные улучшения в руки пользователей, поэтому им необходимо знать, над какими возможностями они будут работать на каждом этапе.

Простое управление изменениями

Процессы разработки ПО должны быть настолько просты, насколько возможно для того, чтобы работа была сделана отлично, но не проще. Пассивный контроль не годится для новых проектов, которые требуют частых изменений. Сделайте процесс внесения изменений «обтекаемым», чтобы минимальное количество людей максимально быстро принимали решения об изменении требований. Это не означает, что каждый разработчик должен просто вносить любые изменения, которые нравятся ему или клиентам: это приведет к хаосу, а не убыстрит разработку.

Что теперь?

Если вы обслуживаете продукт, применяете готовые решения, обращаетесь к сторонним разработчикам или создаете проект «с нуля», изучите приемы формулирования требований, описанные в главе 3, чтобы выбрать из них тот, который сделает ваш проект ценным и успешным. Если у вас возникали связанные с требованиями проблемы, когда вы прежде занимались разработкой проектов таких типов, как те, что описаны в этой главе, проанализируйте их — это поможет диагностировать проблемы и определить основные причины. Если вы воспользуетесь приемами из главы 3 или способами, описанными здесь, то удастся ли вам предотвратить возникновение проблем в следующем проекте подобного типа?

Глава 17 От разработки требований — к следующим этапам

Работа над Chemical Tracking System продвигалась просто замечательно. Но спонсор проекта Жерар и сторонник продукта от персонала, работающего на складе, Роксана, сомневались, стоит ли тратить много времени на определение требований, Тем не менее они присоединились к разработчикам и другим сторонникам продукта, которые проводили однодневный тренинг, посвященный работе над требованиями к ПО. На этом тренинге подчеркивалась важность достижения общего понимания всеми заинтересованными в проекте лицами бизнес-целей и нужд пользователей. Кроме того, всех участников ознакомили с терминологией, касающейся требований, концепциями и приемами, которые будут использоваться в работе. А также призвали применять лучшие приемы для работы с требованиями на практике.

По мере развития проекта Жерар получил отличные отзывы от представителей пользователей о том, насколько хорошо прошла разработка требований. Он даже организовал ланч для аналитиков и апологетов продукта, чтобы отпраздновать создание основной версии требований для первого выпуска системы. На ланче Жерар поблагодарил тех, кто занимался сбором информации для создания требований, за их вклад и коллективную работу. А затем сказал: «Теперь, когда с требованиями все в порядке, я с нетерпением ожидаю скорого появления готового кода».

«Мы еще не готовы писать код продукта, — ответил менеджер проекта, — Мы планируем выпускать систему поэтапно, поэтому ее необходимо разрабатывать, сообразуясь с будущими дополнениями. С помощью прототипов мы получили четкое представление о технической стороне и смогли понять характеристики интерфейса, который предпочитают пользователи. Если на этой стадии мы потратим еще немного времени на проектирование, то сможем избежать проблем в течение следующих нескольких месяцев, когда придет время добавлять функциональность к системе».

Жерар расстроился. Все выглядело так, как будто разработчики сознательно медлили, вместо того чтобы прямо приступить к работе. Однако не делал ли он преждевременных выводов?

Опытные менеджеры проекта и разработчики понимают ценность преобразования требований к ПО в надежный дизайн и рациональные планы. Они необходимы в тех случаях, когда следующая версия представляет 1 или 100% всего конечного продукта. В этой главе мы рассмотрим, как преодолеть пропасть, которая отделяет разработку требований от успешного выпуска продукта: несколько вариантов влияния требований на планы проекта, проектирование, написание кода и тестирование, как показано на рис. 17-1.

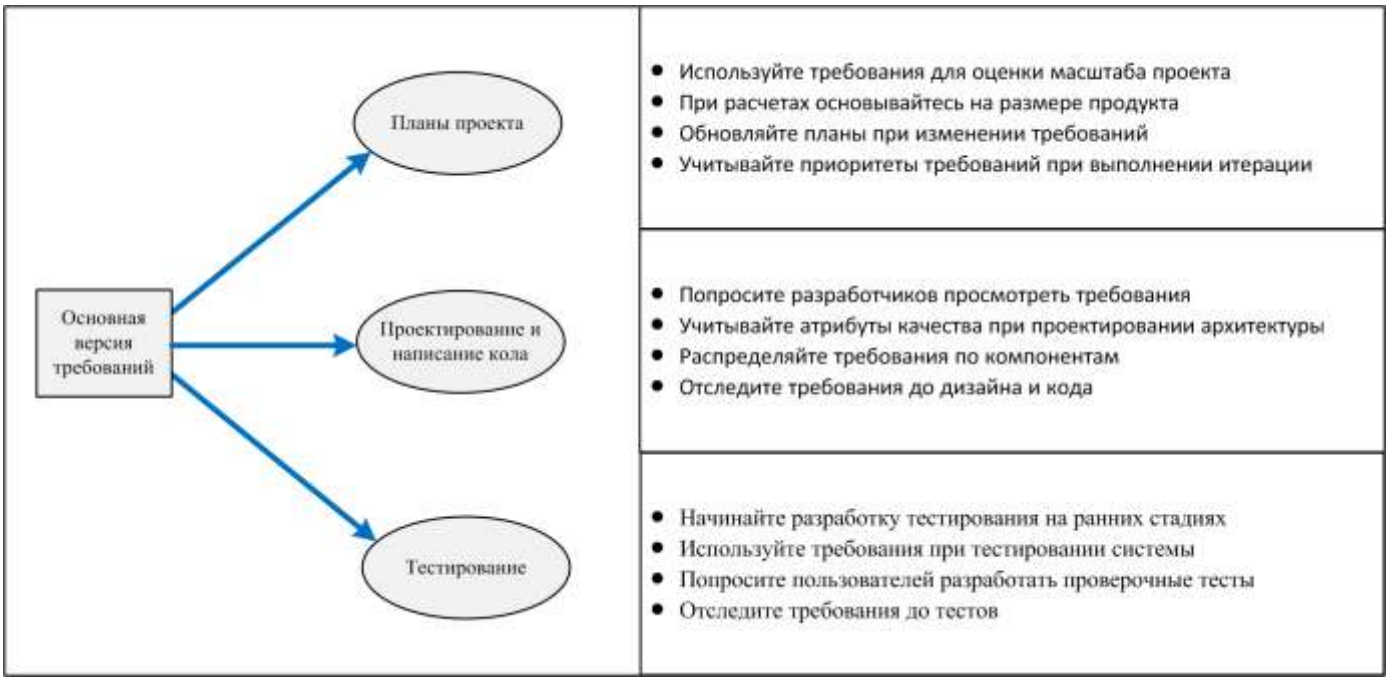


Рис. 17-1. Влияние требований на планирование проекта, дизайн, написание кода и тестирование

От требований к планам проекта

Поскольку именно требования определяют предполагаемый исход (результат) проекта, планы, сметы и графики следует разрабатывать на основе требований. Однако необходимо помнить, что наиболее важный результат проекта — это та система, которая соответствует бизнес-целям, а не обязательно та, в которой реализованы все первоначально требования согласно первоначальному плану проекта.

В требованиях и планах указана первоначальная оценка затрат на проект, определенная членами команды. Однако границы проекты могут выйти за первоначальные рамки, да и планы могут оказаться невыполнимыми. Кроме того, бизнес-потребности, бизнес-правила и ограничения проекта могут измениться. Успех проекта сомнителен, если вы не будете обновлять ваши планы в соответствии с изменяющимися целями и обстоятельствами.

- Менеджеров проекта часто интересует, сколько времени и усилий понадобится для разработки требований. Для небольших проектов, которыми обычно занималась наша команда, этот этап обычно стоил от 12 до 15% всех затрат (Wiegers, 1996a), однако показатель зависит от объема и сложности проекта. Несмотря на опасения, что работа над требованиями замедлит создание продукта, доказано, что понятные требования ускоряют процесс разработки, что и показывают следующие примеры;
- изучение 15 проектов в сфере телекоммуникаций и банковской сферы показало, что в наиболее успешных проектах примерно 28% ресурсов тратилось на разработку требований (Hofmann и Lehner 2001): 11% — на сбор информации по требованиям, 10% — на моделирование, а 7% — на проверку и утверждение. На разработку требований для среднего проекта необходимо 15,7% всех ресурсов и 38,6% времени;
- в проектах NASA, в которых затрачивалось более 10% всех ресурсов на разработку требований, затраты и отклонения от графика оказались существенно ниже, чем в проектах, где на требования затрачивалось меньше ресурсов (Hooks и Farry, 2001);
- исследования, проведенные в Европе, показали, что команды, разрабатывающие продукты более быстро, посвятили больше времени и усилий требованиям, чем более медленные команды (табл. 17-1) (Blackburn, ScuddernVanWassenhove, 1996).

Не все ресурсы на разработку требований должны расходоваться в начале проекта, как в модели «водопада» или последовательной модели жизненного цикла. В итеративных моделях определенное время на требования будет тратиться в каждом повторном цикле разработки. Цель таких проектов — реализовывать функциональность каждые несколько недель, поэтому

требованиями придется заниматься часто, но мало. На рис. 17-2 показано, как в разных моделях жизненного цикла в период разработки продукта распределяются усилия на требования.

Таблица 17-1. Затраты на требования ускоряют разработку

	Усилия, затраченные на требования	Время, затраченное на требования
Более быстрые проекты	14%	17%
Более медленные проекты	7%	9%

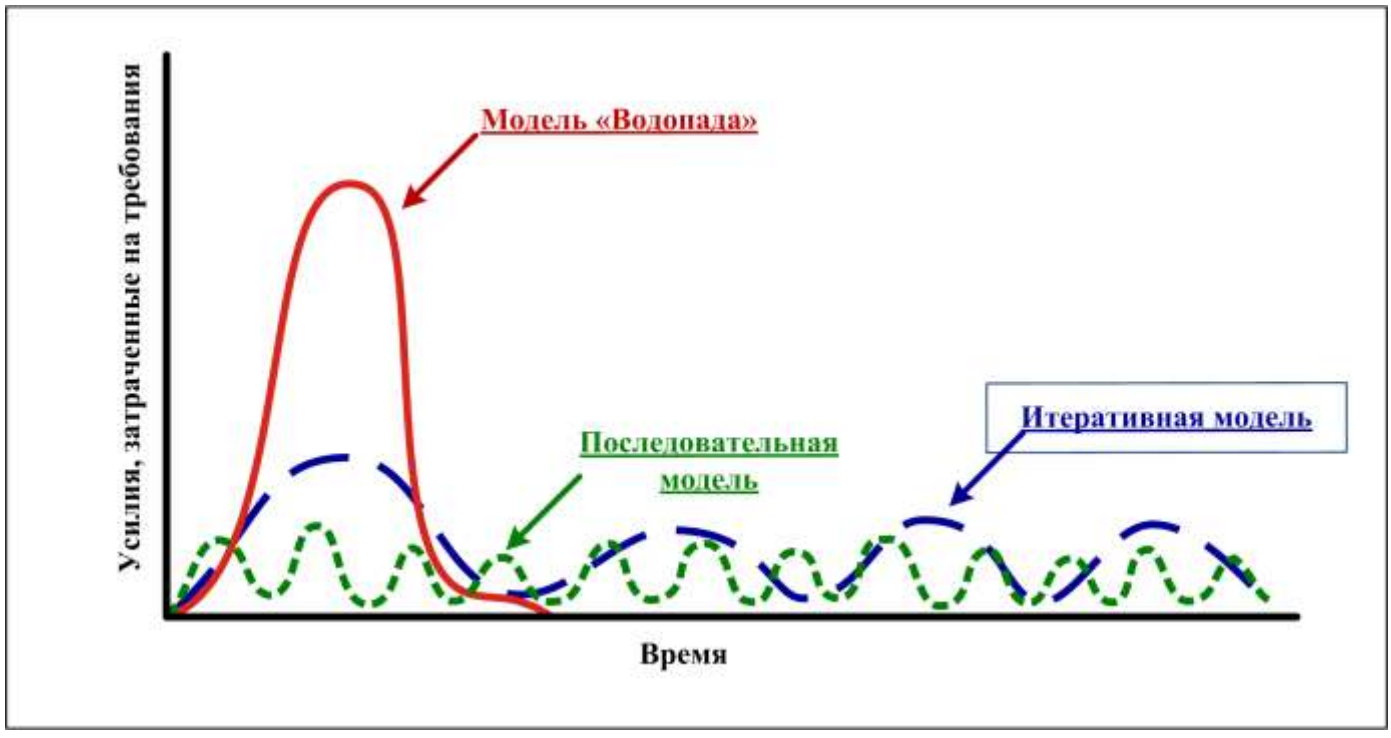


Рис. 17-2. Распределение затрат на требования по времени различается для проектов, разрабатываемых на основе различных моделей жизненного цикла

Ловушка

Старайтесь избегать паралича аналитического процесса. Если в самом начале проекта масса усилий тратится на разработку совершенных и полных; требований — «раз и навсегда», то зачастую мало полезной функциональности удается реализовать в срок. С другой стороны, не следует избегать разработки требований вообще только из-за боязни паралича аналитического процесса.

Требования и расчеты

Первый шаг при оценке проекта — прикинуть объем продукта ПО. Это делают исходя из текстовых требований, моделей анализа, прототипов или элементов пользовательского интерфейса. Хотя не существует идеального мерила размера ПО, чаще всего используются следующие:

- количество отдельно тестируемых требований (Wilson, 1995);
- функциональные точки и характерные точки (Jones, 1996b) или трехмерные функциональные точки, включающие в себя данные, функции и элементы управления

(Whitmire, 1995);

- количество, тип и сложность элементов ***графического пользовательского интерфейса*** (graphical user interface, GUI);
- оценка строк кода, необходимого для реализации специальных требований;
- количество классов объектов или других объектно-ориентированных метрик (Whitmire, 1997).

Все эти методы годятся для расчета размера. Выбирайте любой, учитывая собственный опыт и природу разрабатываемого ПО. Понимание того, что команда добилась успеха при разработке схожих проектов, используя аналогичные технологии, поможет вам оценить результативность работы команды. Как только вы получите данные о размере, вы сможете воспользоваться коммерческим инструментом для расчета, который предлагает допустимые комбинации затрат на разработку и сроков. Эти средства позволят вам привести в порядок расчеты, выполненные на основе таких факторов, как квалификация разработчиков, сложность проекта, а также опыт команды в данной предметной области. Информация о некоторых доступных средствах оценки ПО опубликована на *http://www.laatuk.com/tools/effort_estimation_tools.html. (Сайт закрыт прим. Редактора)*

Если вы не сравните ваши расчеты с действительными результатами проекта и не улучшите методы расчета, то ваша оценка вечно будет оставаться всего лишь предположением. Чтобы собрать достаточно данных для корреляции размера ПО с усилиями на его разработку, потребуется время. Ваша цель — вывести уравнения, которое позволит с уверенностью оценить размер завершенного ПО на основании требований к нему.

Однако даже самые лучшие методики не сработают, если клиенты, менеджеры или юристы будут часто менять требования. Если изменения настолько велики, что аналитик и разработчики не успевают их учитывать, команда может впасть в ступор, не имея возможности продолжать проект. В этом случае, возможно, стоит приостановить работу, пока потребности клиентов не прояснятся.

Определенные крайние сроки поставки — головная боль для менеджеров большинства проектов. Каждый раз, когда заданные сроки и тщательно просчитанный график не совпадают, необходимы переговоры. Менеджер проекта, который может аргументировать расчеты, тщательно выполненные и основанные на фактах, выглядит при переговорах гораздо лучше, чем оперирующий предположениями. Если лица, заинтересованные в проекте, не могут разрешить конфликт, разгоревшийся вокруг графика, стоит обратить внимание на бизнес-цели проекта: они будут определять, стоит ли уменьшить объем, добавить ресурсы или согласиться на некоторое снижение качества. Такие решения принимать нелегко, однако это единственный способ максимально повысить ценность подлежащего сдаче продукта.

Есть час?

Как-то клиент попросил наших разработчиков адаптировать небольшую программу, которую он написал для себя, к нашей общей компьютерной сети, чтобы и его коллеги могли ее использовать. «Есть час?» - спросил меня наш менеджер, передавая мне поверхностную оценку размера проекта. Когда я порасспросил клиента и его коллег, чтобы понять, что же им требуется, оказалось, что проблем несколько больше. Я потратил 100 часов только на написание программы, которая им была нужна, без наведения лоска. Тот факт, я потратил в 100 раз больше времени, чем рассчитывал менеджер, говорит о том, что его расчеты были несколько поспешными и основывались на неполной информации. Аналитик должен изучить требования, оценить масштаб и составить представление о размере до того, как кто-нибудь приступит к расчетам или возьмет на себя определенные обязательства.

Неясно сформулированные требования неизбежно приводят к неточным расчетам размера, затрат и графика. Поскольку определенные неясности в требованиях на ранних стадиях проекта

неизбежны, заложите в график и бюджет резерв для непредвиденных обстоятельств, чтобы учесть некоторое разрастание требований (Wiegers, 2002d). Оно происходит из-за изменения ситуации в бизнесе, смещения приоритетов пользователей и рынка, а также того, что со временем заинтересованные в проекте лица лучше начинают понимать, что же продукт может и должен делать. Однако значительное увеличение требований, как правило, указывает на то, что масса положений была упущена в ходе сбора информации.

Отведите время на выбор соответствующих измерений для тех типов проектов, над которыми работает ваша команда. Между размером продукта, затратами, сроком разработки, производительностью и временем становления команды существуют сложные нелинейные взаимоотношения (Putnam и Myers, 1997). Если вы разберетесь в них, вы сможете избежать «области невозможного», то есть такой комбинации факторов, зависящих от размера продукта, графика и сотрудников, при которой еще не один продукт не был успешно разработан.

Ловушка

Не позволяйте, чтобы на ваши расчеты влияли желания других. Ваши прогнозы не должны меняться только потому, что кому-то они не нравятся. Слишком большое несоответствие в прогнозах указывает на необходимость переговоров.

Требования и график

В некоторых проектах применяется график «справа - налево»: сначала устанавливают дату поставки продукта, а затем определяют требования. В таком случае разработчики зачастую часто не успевают закончить работу в указанные сроки, включив в ПО всю необходимую функциональность соответствующего уровня. Более реалистично определить требования к ПО до составления подробных планов и принятия определенных обязательств. Тем не менее стратегия «проектирования по расписанию» может сработать, если менеджеру проекта удастся договориться с заказчиком, какая часть желаемой функциональности удовлетворит его в указанные сроки. Как и всегда, расстановка приоритетов требований — ключевой фактор для успеха проекта.

Для сложных систем, в которых ПО является лишь частью конечного продукта, детальные графики, как правило, составляются после разработки требований на уровне продукта (системы) и предварительного определения архитектуры. На этом этапе ключевые даты поставки могут быть определены и согласованы на основе информации из таких источников, как отдел маркетинга, отдел продаж, отдел по работе с клиентами и разработчики.

Рассмотрите возможность поэтапного планирования и финансирования проекта. На стадии первоначального изучения требований вы получите достаточно информации, чтобы составить реалистичные планы и расчеты для одного или более этапов сборки. Проекты, требования к которым сформулированы нечетко, выиграют в случае поэтапного цикла разработки. Эта модель позволит разработчикам приступить к созданию качественного ПО задолго до полного выяснения требований. Расставив приоритеты требований, вы сможете определить очередность включения функциональности на каждом этапе.

Часто причина неудачи проекта по разработке ПО кроется в том, что разработчики и другие участники проекта плохо составляют планы, а не в том, что они плохие специалисты. К главным ошибкам планирования относятся пропуск стандартных задач, недооценка затрат или сроков, а также проектных рисков, необоснованный оптимизм. Кроме того, не забудьте учесть возможные переделки. Эффективное планирование проекта предполагает наличие следующих элементов:

- оценку размера продукта;
- информацию о производительности специалистов, основанную на прошлом опыте;
- список задач, которые необходимы для полной реализации или проверки функции или варианта использования;
- требования с приемлемым уровнем стабильности;
- опыт, который позволит менеджеру проекта учесть нематериальные факторы и

индивидуальные особенности каждого проекта.

Ловушка

Не поддавайтесь давлению и не принимайте на себя невыполнимых обязательств. Это верный путь к неудаче.

От требований — к проектированию и коду

Граница между требованиями и проектированию не четкая, однако постарайтесь не делать в ваших спецификациях упор на реализацию, кроме тех случаев, когда имеется веская причина для намеренного ограничения проектирования. В идеале на описание предполагаемых функций системы не должно влиять представление о проектировании (Jackson, 1995). Надо признать, что во многих проектах учитываются ограничения проектирования, налагаемые разработанными ранее продуктами, а совместимость с предыдущими продуктами — самое стандартное требование. Именно поэтому в спецификации к требованиям практически всегда содержится некоторая информация о проектировании. Однако в спецификации не должны содержаться случайные элементы проектирования — то есть ненужные или незапланированные ограничения конечной архитектуры. Подключите проектировщиков или разработчиков к экспертизе требований, чтобы удостовериться, что требования могут служить надежной основой для проектирования. Требования к продукту, атрибуты качества и характеристики пользователей влияют на архитектуру продукта (Bass, Clements и Kazman, 1998). Изучая предлагаемую архитектуру, вы рассмотрите разные точки зрения, что поможет вам проверить требования и отшлифовать их точность, как и при использовании прототипов. Оба метода базируются на следующем посыле: «Если я правильно понимаю требования, то способ, который я сейчас проверяю, позволит удовлетворить их. Теперь же, когда у меня в руках предварительная архитектура (или прототип), поможет ли это мне лучше разобраться в требованиях?»

Архитектура особенно важна для систем, в которые входят компоненты ПО и оборудования, а также для сложных систем, состоящих исключительно из ПО. Важный этап анализа требований — распределение высокоуровневых требований к системе по различным подсистемам и компонентам. Системный инженер, аналитик или архитектор классифицирует требования к системе по их принадлежность к функциональным и требованиям к оборудованию (Leffingwell и Widrig, 2000). Трассируемая информация позволяет разработчикам отследить, в каких элементах архитектуры будет отражено каждое требование.

В результате неверного распределения решений может статься, что ПО, определенное как функциональное, следовало бы назначить компонентам оборудования (или наоборот); кроме того, в таком случае возможна низкая производительность или трудности при замене некоего компонента для улучшения версии. При разработке одного проекта инженер по оборудованию сообщил моей группе, что в нашем ПО должны быть преодолены все ограничения, налагаемые его архитектуре оборудования! Хотя ПО само по себе более гибко, чем оборудование, инженерам не следует использовать это качество, чтобы сэкономить на архитектуре оборудования. Воспользуйтесь приемом проектирования и создания программ, чтобы выработать оптимальные решения того, какие возможности каждый системный компонент будет удовлетворять в рамках заданных ограничений.

Распределение системных функций по подсистемам и компонентам надо выполнять «сверху вниз» (Hooks и Farry, 2001). Представьте DVD-проигрыватель, в который входит мотор, открывающий и закрывающий лоток дисковода и вращающий диск, оптическая подсистема для считывания данных с диска, ПО для формирования изображений, многофункциональный пульт дистанционного управления и многое другое, как показано на рис. 17-3. Подсистемы взаимодействуют для управления поведением в таких случаях, когда, скажем, пользователь нажимает кнопку на пульте дистанционного управления, чтобы открыть лоток дисковода во время проигрывания диска. В таких сложных продуктах требования к системе влияют на архитектуру, а

архитектура в свою очередь влияет на распределение требований.

При разработке некоторых проектов архитектура ПО немного изменяется, тем не менее на проектирование время никогда не бывает потрачено зря. Разнообразные разработки ПО удовлетворяют большинству требований к продукту. Они различаются по производительности, эффективности и устойчивости к сбоям, а также по использованным техническим методам. Если вы перескакиваете от требований к коду, это означает, что вы по существу разрабатываете ПО «на лету». Вы предлагаете *какую-то* архитектуру, но не обязательно *идеальную,* вероятный результат в этом случае — плохо структурированное ПО. Переделывая код вы, конечно же, улучшите архитектуру (Fowler, 1999). Однако унция заранее разработанного дизайна стоит фунта переделок уже выпущенного продукта. Продумав альтернативы архитектуры, вы будете уверены, что разработчики будут принимать во внимание любые указанные ей ограничения.

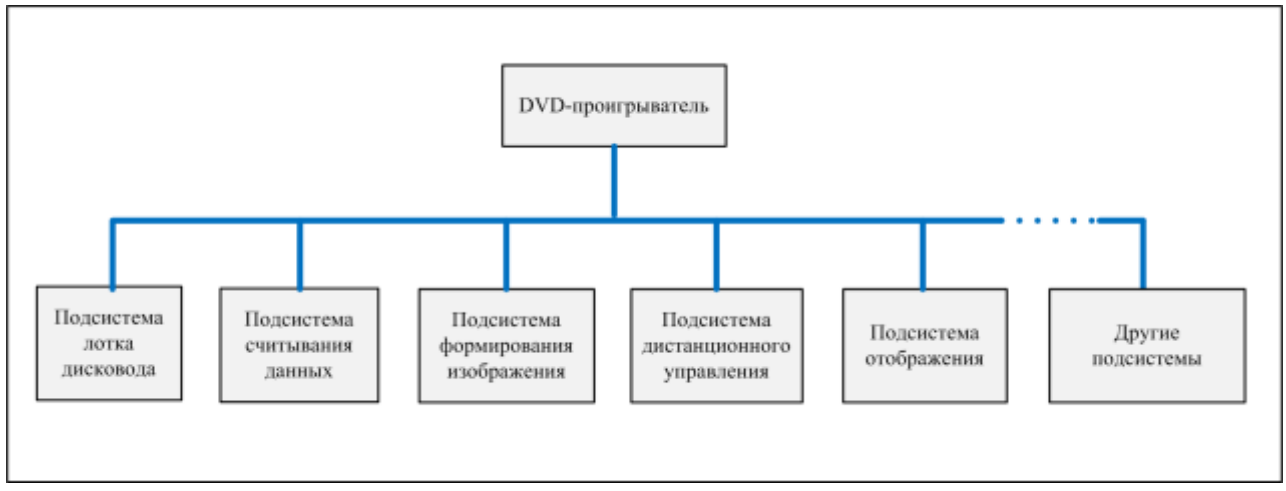


Рис. 17-3. Такие сложные продукты, как DVD-проигрывателе, содержат множество подсистем ПО и оборудования

Невероятно уменьшившийся проект

Как-то я занимался проектом, имитирующим поведение фотографической системы с помощью восьми вычислительных процессов. После скрупулезного анализа требований команде не терпелось приступить к написанию кода. Но вместо этого нам пришлось создавать модель — в тот момент мы больше думали о создании решения, а не пытались разобраться в проблеме. Мы быстро поняли, что в трех этапах моделирования использовались идентичные вычислительные алгоритмы, в трех других применялся еще один набор алгоритмов, а в остальных двух — третий набор. Взгляд с позиций проектирования позволил упростить проблему и уменьшить число сложных вычислений с восьми до трех. Начни мы кодировать сразу же после анализа требований, в какой-то момент мы бы без сомнения обнаружили повторение кода, но мы сэкономили массу времени, выявив их гораздо раньше. Изменение моделей проектирования намного более эффективно, чем переделка кода!

Как и в случае с требованиями, отличный дизайн — это результат итераций. Многократно прорабатывайте проектные решения для уточнения первоначальной концепции по мере получения информации и появления дополнительных идей. При недостатках в архитектуре появляются продукты, которые трудно обслуживать и расширять, которые не удовлетворяют пожеланиям клиентов в отношении производительности, легкости и простоты использования и надежности. Время, затраченное на преобразование требований в проектные решения, — это отличная инвестиция в создание высококачественных и надежных продуктов.

Вам не нужно разрабатывать полный подробную архитектуру всего продукта до начала реализации, однако следует выполнить архитектуру каждого компонента до того, как вы приступите к ее реализации. Планирование архитектуры наиболее полезно для особенно трудных проектов, где разрабатываются системы со множеством внутренних граничащих и взаимодействующих друг с другом компонентов, а также тех, которыми занимаются неопытные разработчики (McConnell, 1998). Однако проекты любого типа выиграют, если вы:

- разработаете надежную архитектуру подсистем и компонентов, которая выдержит будущие изменения;
- определите ключевые классы объектов или функциональные модули, которые необходимо создать, определив их интерфейсы, функции и взаимодействия с другими элементами;
- разберетесь, как выполняются планируемые потоки или где размещается функциональность в параллельных процессах для параллельно работающих систем;
- определите предполагаемую функциональность каждой единицы кода, которая следует основному принципу архитектуры: сильная связанность, слабая связь и сокрытие информации (McConnell, 1993);
- убедитесь, что в архитектуру включены все функциональные требования и что она не содержит ненужной функциональности;
- удостоверитесь, что архитектура учитывает все условия исключения, которые могут возникнуть;
- убедитесь, что архитектура будет отвечать заданным критериям производительности, устойчивости к сбоям, надежности и др.

По мере преобразования требований в архитектуру и код разработчики сталкиваются с определенными неясными и запутанными местами. В идеале они могут обратиться к клиентам или

аналитиками за разъяснениями. Если же проблему нельзя решить немедленно, любые предположения, догадки или интерпретации разработчики должны задокументировать и отправить на просмотр представителям клиента. Если таких проблем много, значит, требования сформулированы недостаточно ясно и детально до того, как аналитик передал их разработчикам. Просмотрите остальные требования вместе с одним или несколькими разработчиками и уточните их, прежде чем продолжите сборку продукта. Также пересмотрите процессы утверждения требований, чтобы выяснить, как получилось так, что требования низкого качества попали к разработчикам.

От требований — к тестированию

Тестирование и разработка требований связаны синергетически. Согласно мнению консультанта Dorothy Graham: «Чем лучше требования, тем качественнее тесты; чем качественнее анализ тестирования, тем лучше требования» (Graham, 2002). Требования обеспечивают основу для тестирования системы. Продукт следует тестировать на соответствие тому, что он, как записано в требованиях, должен делать, а не на предмет архитектуры или кода. Тестирование системы, выполненное на основе кода, может стать накликанной бедой. Продукт может вести себя именно так, как описано в вариантах тестировании, созданных на основе кода, но это не означает, что он соответствующим образом реализует пользовательские или функциональные требования. Подключите тестеров к экспертизе требований, чтобы убедиться, что требования поддаются проверке и могут служить основой для тестирования системы.

Что тестировать?

Участник семинара как-то заметил: «Я работаю в группе тестирования. У нас нет написанных требований, поэтому мы должны протестировать систему на предмет того, что она с нашей точки зрения должна делать. Иногда мы ошибаемся, тогда мы узнаем у разработчиков, каковы функции системы, и тестируем ее снова». Тестирование того, что разработчики создали, — не то же самое, что тестирование того, что они должны были создать. Требования — это конечный документ для системных и пользовательских проверочных испытаний. Если требования к системе сформулированы плохо, то тестеры обнаружат множество предположений, реализованных разработчиками. Некоторые из них отражают соответствующие решения разработчиков, другие же отшлифованы или неверно истолкованы. Аналитик должен включить предположения и их источники в спецификацию требований к ПО, чтобы провести будущее повторное тестирование более эффективно.

Тестеры проекты должны определить, как они будут проверять каждое требование. Возможных методов несколько:

- тестирование (выполнение ПО с целью поиска дефектов);
- экспертиза (проверка кода на предмет его соответствия требованиям);
- демонстрация (демонстрация того, что продукт работает, как ожидалось);
- анализ (обоснование того, как система должна работать при определенных условиях).

Само по себе обдумывание того, как проверить каждое требование, является ценным приемом работы над качеством. Используйте такие приемы анализа, как графики типа «причина-результат», чтобы составить варианты тестирования на основе логики, описанной в требовании (Myers, 1979). При этом обнаруживаются неясности, пропуски или предположения и другие проблемы. Необходимо, чтобы каждое функциональное требование можно было проследить по крайней мере до одного варианта тестирования в системном тестовом наборе, для того чтобы ни одна ожидаемая реакция системы не осталась непроверенной. Вы можете измерить прогресс тестирования по частям, высчитывая процент требований, которые прошли проверку. Опытные тестеры могут

обогатить тестирование, основанное на требованиях, дополнительными тестами, основанными на истории продукта, предполагаемых сценариях использования, общих характеристиках качества и случайностях.

В тестировании на основе требований применяются различные стратегии тестирования: зависящие от выполняемых действий, отданных (включая анализ значений границ и разбиения на эквивалентные классы), от логики от событий и от состояния (Poston, 1996). Можно генерировать варианты тестирования автоматически, используя официальную спецификацию, однако если вы будете использовать более распространенные спецификации к требованиям, составленные на естественном языке, вам придется разрабатывать варианты тестирования вручную.

По мере продвижения разработки команда будет конкретизировать требования от высоких уровней абстракции, как, например, в вариантах использования, через детализированные функциональные требования к ПО к спецификациям для отдельных модулей кода. Авторитетный специалист в области тестирования Boris Beizer (1999) подчеркивает, что тестирование на основе требований должно выполняться на каждом уровне конструирования ПО, а не только на конечном пользовательском уровне. Масса кода в приложении не доступно пользователям напрямую, однако он необходим для внутренних взаимодействий. Каждый модуль должен удовлетворять собственной спецификации, даже если функция этого модуля невидима для пользователя. Как следствие, тестирование системы на основе пользовательских требований — необходимая, но не достаточная стратегия тестирования системы.

От требований — к успеху

Недавно я наблюдал такую ситуацию: разработчики-контрактники присоединились к проекту для реализации приложения, требования для которого писала друга команда. Новички только взглянули на дюжину трехдюймовых переплетов документации по требованиям, вздрогнули от ужаса и начали программировать. В ходе сборки они не обращались, к спецификации. Вместо этого они построили то, что, по их мнению, задумывалось, основываясь на неполном и неточном понимании будущей системы. Неудивительно, что возникло множество проблем. Перспектива разбираться в массе даже самых совершенных требований без сомнения приводит в уныние, но игнорирование требований — это верный путь к провалу проекта. Гораздо быстрее прочитать требования, какими бы объемными они не были, до реализации, чем создавать ненужную систему, а затем переделывать ее. Еще лучше собрать разработчиков в начале проекта, чтобы они смогли поучаствовать в работе над требованиями и уже тогда создать прототипы.

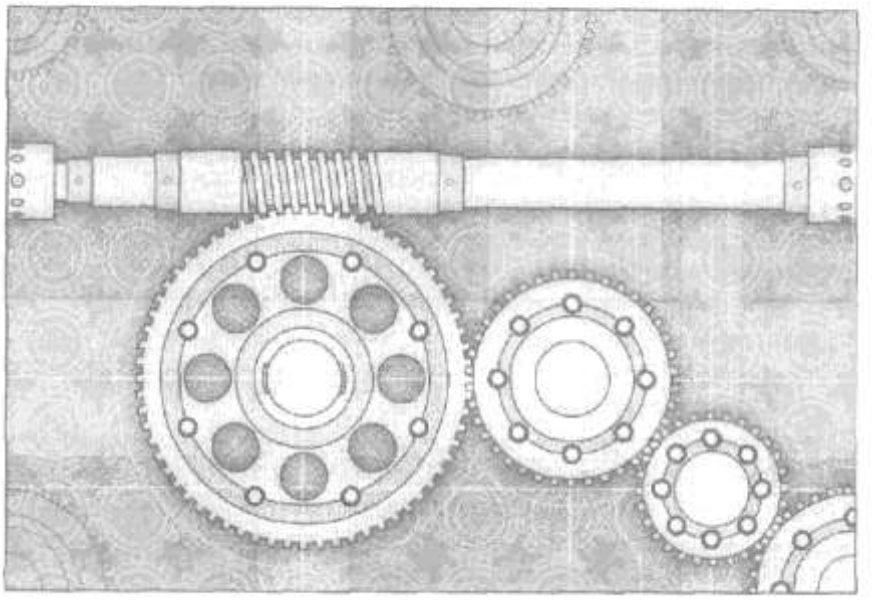
Практика более успешного проекта такова: составлялся список всех требований, которые были включены в определенную версию. Группа ***контроля качества проекта*** (quality assurance, QA) оценивала каждую версию, тестируя все требования. То из них, которое не удовлетворяло критериям тестирования, считалось ошибочным. Группа контроля качества откладывала выпуск версии, если не было удовлетворено количество требований, превышающее заранее оговоренное, или крайне важные требования. В основе успеха проекта лежало использование документированных требований для принятия решения о готовности версии к выпуску,

Конечным, подлежащим сдаче продуктом считается система ПО, отвечающая потребностям и ожиданиям клиента. Требования являются важным этапом на пути от бизнес-потребностей до удовлетворенных клиентов. Если в основе ваших планов проекта, архитектуры ПО и системного тестирования не лежат высококачественные требования, вероятнее всего вам придется затратить много усилий на создание качественного продукта. Однако не становитесь и рабом требований. Не имеет смысла тратить массу времени на создание ненужной документации и проведение ритуальных встреч, если при этом не создается ПО и проект закрывается. Попытайтесь достичь разумного баланса между доскональной спецификацией и кодированием «из головы» — это снизит до приемлемого уровня риск создания ненадлежащего продукта.

Что теперь?

- Попробуйте отследить все требования в реализованной части спецификации требований к ПО до отдельных элементов дизайна. Это можно отразить в моделях потока данных, таблице в диаграмме «сущность-связь», объектных классах или методах или других элементах дизайна. Недостающие элементы дизайна могут указывать на то, что разработчики выполнили всю свою работу в уме, перейдя сразу от требований к коду. Если разработчики нерегулярно занимаются дизайном, прежде чем приступить к написанию кода, возможно, их следует обучить дизайну ПО.
- Запишите количество строк кода, функциональных точек, классов объектов или элементов пользовательского интерфейса, которые необходимы для реализации каждой функции продукта или варианта использования. Укажите вашу оценку затрат, необходимых для полной реализации и проверки каждой функции или варианта использования, а также реальные затраты. Выявите взаимосвязи размера продукта и затраченных усилий — это поможет вам выполнить более точные расчеты для будущих спецификаций к требованиям.
- Рассчитайте средний процент незапланированного роста проекта для нескольких последних проектов. Можете ли вы включить резерв для непредвиденных обстоятельств в график проекта для учета такого увеличения объема в будущих проекта? Используйте данные об увеличении объема из предыдущих проектов для объяснения этого резерва, чтобы он не выглядел как произвольное решение для менеджеров, клиентов и других заинтересованных лиц.

Часть III. Управление требованиями к ПО



Глава 1 Принципы и приемы управления требованиями к ПО

В главе 1 говорилось, что разработку технических условий можно разделить на разработку требований и управление требованиями. Стадия разработки подразумевает сбор информации, анализ, уточнение и утверждение требований. Как правило, она заканчивается созданием пакета документов; документа об образе и границах продукта, документации к вариантам использования, спецификации требований к ПО, словаря данных и соответствующих моделей анализа. После проверки и одобрения этот пакет определяет базовую версию требований, соглашение между разработчиками и клиентами. Вероятно, в ходе разработки заключаются и дополнительные соглашения, касающиеся готового продукта, ограничений, графиков, бюджета или контрактных обязательств; но эти темы выходят за рамки данной книги.

Соглашение по требованиям является связующим звеном между разработкой и управлением требованиями. У членов команды должен быть доступ к текущим требованиям на протяжении всего проекта, возможно, через Web-решения посредством инструментов управления требованиями. Под управлением требованиями подразумевают все действия, поддерживающие целостность, точность и своевременность обновления соглашения о требованиях в ходе проекта. Как показано на рис. 18-1, это:

- управление изменениями базовой версии требований;
- поддержание планов проекта актуальными в соответствии с меняющимися требованиями;
- управление версиями отдельных требований и документации требований;
- контроль состояния требований в базовой версии;
- управление логическими связями между отдельными требованиями и другими материалами для работы с проектом.

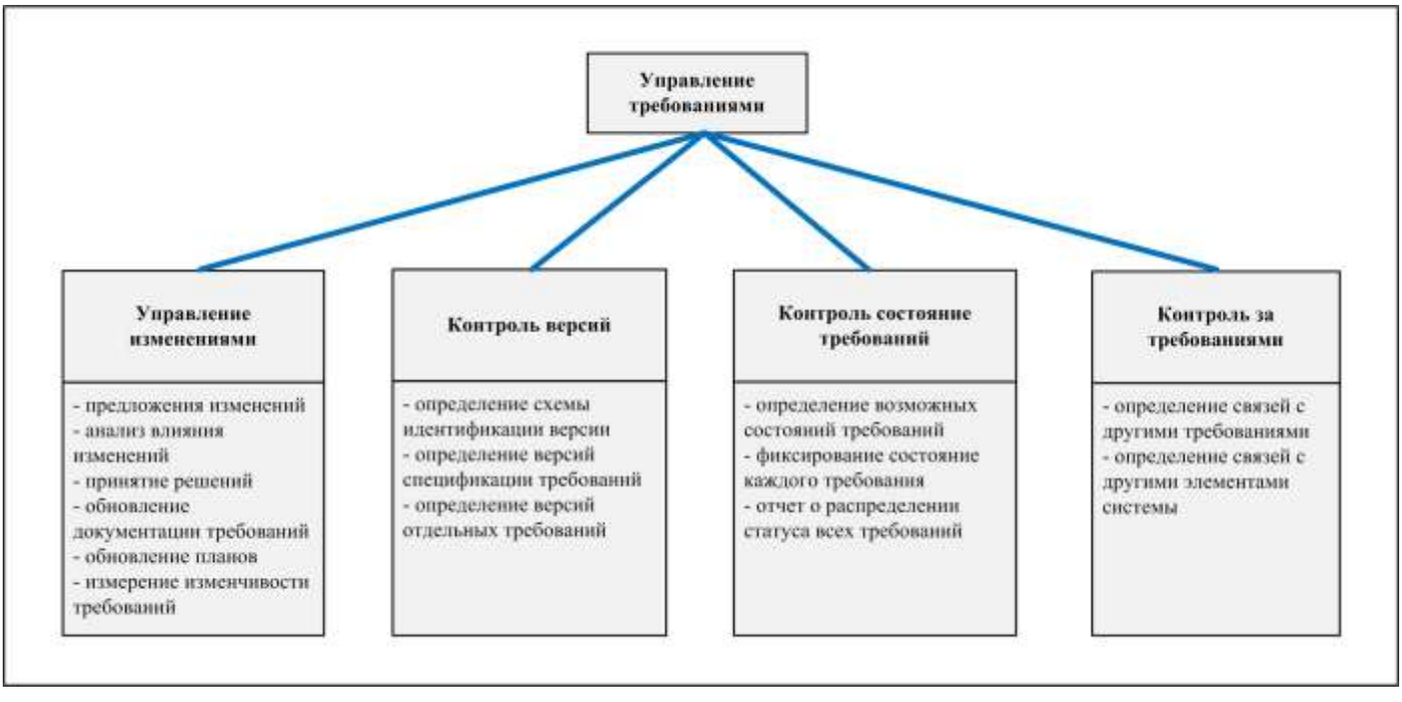


Рис. 18-1. Основные составляющие управления требованиями

Разработчики, принимающие предлагаемые изменения требований, не всегда способны соблюдать текущий график и выполнять обязательства, касающиеся качества. Менеджер проекта должен обговорить изменения этих обязательств с другими менеджерами, работающими над проектом, клиентами и всеми заинтересованными в проекте лицами. Введение новых или изменение существующих требований 'влияет на проект следующим образом:

- откладывается реализация требований низших уровней;

- приглашаются новые специалисты;
- на короткий период времени вводится режим сверхурочной работы, по возможности оплачиваемой;
- в соответствии с новой функциональностью изменяется график;
- страдает качество, так как необходимо уложиться в график (слишком часто это реакция по умолчанию).

Ни один из этих способов нельзя назвать идеальным для всех случаев, поскольку гибкость проектов зависит от функций, опыта специалистов, бюджета, графика и качества (Wiegers, 1996a). При принятии решений полагайтесь на приоритеты, установленные при планировании проекта ключевыми фигурами из тех, кто заинтересован в проекте. Независимо от того, какова будет ваша реакция на корректировку требований, при необходимости примите как данность изменения ожиданий и обязательств. Это гораздо лучше, чем надеяться, что каким-то образом к принятой в начале проекта дате выпуска все новые функции появятся в продукте, причем без таких последствий, как перерасход бюджета или переутомление команды.

В этой главе рассматриваются основные принципы управления требованиями. В других главах части III некоторые приемы управления требованиями описываются более подробно, в том числе изменение управлением (глава 19), анализ влияния изменений (также глава 19) и контроль за требованиями (глава 20). Завершается часть III обсуждением коммерческих средств, с помощью которых команда управляет требованиями (глава 21).

Базовая версия требований

Базовая версия (baseline) — это набор функциональных и нефункциональных требований, которые разработчики обязались реализовать в определенной (выбранной) версии. Определение базовой версии позволяет заинтересованным в проекте лицам выработать общее понимание возможностей и свойств, которые они хотят видеть в данной версии. Принятая базовая версия требований — как правило, версия становится базовой после официальных экспертизы и утверждения — передается для управления конфигурацией. Последующие изменения разрешается вносить в нее только через определенный в проекте процесс управления изменениями. До принятия базовой версии требования все еще изменяются, поэтому не имеет смысла ограничивать модификацию какими-то процедурами. Однако начинайте контролировать версии, уникальным образом идентифицируя разные версии каждого документа требований, сразу после того, как сделаете предварительный набросок этого документа,

С практической точки зрения, положения в базовой версии должны отличаться от предложенных, но не принятых требований. Необходимо, чтобы основная версия спецификации требований к ПО содержала только те требования, которые запланированы для конкретного выпуска. Этот документ должен быть ясно определен как базовый, чтобы отличать его от серии выполненных ранее набросков версий, которые еще не утверждены. Сохранение требования в средстве управления требованиями облегчает идентификацию требований, принадлежащих определенной базовой версии, и управление изменениями в ней. В этой главе описываются некоторые способы управления требованиями в базовой версии.

Процедуры управления требованиями

Необходимо определить действия, которые, как ожидается, проектная команда будет выполнять для управления требованиями. Задокументируйте эти действия и организуйте тренинг для специалистов, чтобы сотрудники организации могли выполнять их последовательно и эффективно. Обратите внимание на следующее:

- на инструменты, приемы и соглашения для управления версиями
- различных документов требований и отдельных требований;
- на то, как составляется базовая версия требований;
- на статус требований, которые будут использоваться, и лица, которые имеют право изменять их;
- на процедуры контроля за статусом требования;
- на способы, с помощью которых новые требования и изменения существующих требований предлагаются, обрабатываются, обсуждаются и передаются всем

заинтересованным лицам;

- на методы анализа влияния предложенного изменения;
- на то, как на планах и обязательствах проекта отразятся изменения требований.

Вы можете включить всю эту информацию в одно описание процесса управления требованиями. Или же разработать отдельные процедуры для контроля за изменениями, анализа влияния и контроля за состоянием. Эти процедуры следует довести до сведения каждого сотрудника организации, поскольку они представляют общие функции, которые должны выполняться каждым проектной командой.

Дополнительная информация

В главе 22 описано несколько полезных образцов документов для управления требованиями.

Кто-то должен выполнять действия при управлении требованиями поэтому в описаниях процесса также следует определить роли участников, ответственных за выполнение каждой задачи. Проектный аналитик требований, как правило, несет основную ответственность за управление требованиями. Он выбирает механизм сохранения требований (например, средство управления требованиями), определяет атрибуты требований, скоординирует обновление данных о состоянии и трассируемости требований и сгенерирует отчеты об изменении действий.

Ловушка

Если никто из участников проекта не несет ответственности за выполнение этапов управления требованиями, не следует ожидать их выполнения.

Контроль версий

«Я наконец доделала функцию запроса по каталогу различных поставщиков, — сообщила Шери на еженедельной встрече, посвященной состоянию проекта Chemical Tracking System. — Вот это была работа!»
«О, клиент отказался от этой функции две недели назад, — подал реплику менеджер проекта Дейв. — Ты разве не получила новую версию спецификации?»
Шери была озадачена. «Как это понимать, отказался? Эти требования черным по белому напечатаны вверху страницы в самой последней версии моей спецификации».
«Мм, их нет в моей копии. У меня версия 1.5 спецификации. А у тебя?»
«У меня тоже 1.5, — с отвращением сказала Шери. — Эти документы должны быть идентичны, но очевидно, это не так. И что, эта функция все еще включена в базовую версию или я потратила впустую 40 часов моей жизни?»

Если вы когда-либо слышали подобный разговор, вы знаете, как расстраиваются люди, когда тратят впустую время, работая над устаревшими или непоследовательными требованиями. Контроль версий — это важный аспект управления спецификациями требований и другими проектными документами. Каждой версии следует присвоить уникальный идентификатор. У каждого члена команды должен быть доступ к текущей версии требований, а изменения необходимо ясно документировать и передавать всем заинтересованным сторонам. Чтобы минимизировать путаницу, конфликты и непонимание, назначьте право обновления требований строго определенным лицам и убедитесь, что идентификатор версии изменяется при каждом изменении требования.

Это не ошибка, а функция!

Разработчики-контрактники как-то получили массу отчетов об ошибках от людей, тестировавших последнюю версию систему, которую они только что поставили клиенту. Разработчики были в недоумении — система успешно прошла все их собственные проверки. После обстоятельного изучения проблемы, выяснилось, что клиенты тестировали новое ПО, сверяясь с устаревшей спецификацией требований. То, что тестеры в отчетах называли ошибками, на самом деле оказалось функциями. (В обычной ситуации это просто шутка, из тех, что любят программисты.) Масса усилий была потрачена впустую, поскольку тестеры неправильно представляли себе поведение системы. Им понадобилось много времени, чтобы переписать тесты в соответствии с последней версией спецификации и повторно протестировать продукт, а все из-за контроля версий.

Каждая версия документа требований должна содержать историю переработки, где указываются внесенные изменения, дата каждого из них, лицо, внесшее изменение, а также причина. Возможно, следует добавлять номер версии к названию каждого отдельного требования, который можно последовательно увеличивать при модификации требований, например **Print.ConfirmCopies-1.**

Простейший механизм управления версиями — именовать вручную каждую версию спецификации требований к ПО согласно соглашению. Попытка различать версии документов по дате изменения или дате печати часто приводит к ошибкам и неразберихе; я не рекомендую их использование. Несколько команд, в которых я работал, успешно применяли такое соглашение: первая версия любого нового документа называется «Версия 1.0, набросок 1». Следующий черновик называется «Версия 1.0, набросок 2». Автор последовательно увеличивает номер наброска при каждом изменении и так до тех пор, пока документ не будет утверждена базовая версия документа. Тогда название меняется на «Версия 1.0, утвержденная». Следующие версии называются «Версия 1.1 набросок 1» при небольшом изменении или «Версия 2.0, набросок 1» при значительном изменении. (Разумеется, термины «небольшой» и «значительный» субъективны или, по крайней мере, зависят от контекста.) При применении этой схемы ясно различаются наброски и версии базового документа, однако необходимо соблюдать строгий порядок в ведении документации.

Если вы сохраняете требования в текстовом редакторе, вы можете отследить коррективы с помощью режима изменений текста. Эта функция выделяет внесенную правку, зачеркивая удаленные фрагменты, подчеркивая добавленные и помечая вертикальными линиями на полях положение каждого изменения. Так как откорректированный документ трудно читать, в текстовом редакторе предусмотрена возможность просмотреть и распечатать документ с пометками или его окончательную версию с принятой правкой. После того как вы составите базовую версию документа с такими пометками, сначала создайте архив версии с пометками, затем примите все коррективы, а затем сохраните очищенную версию как новую базовую версию для следующего раунда изменений.

Более сложный метод предполагает сохранение спецификации требований в таком средстве контроля версий, как тот, что применяется для контроля исходного кода с помощью системы контроля версий (**check-in** и **check-out**). Для этой цели разработана масса коммерческих средств управления конфигурацией. Я знаю об одном проекте, когда несколько сотен написанных в Microsoft Word вариантов использования продукта сохраняется в таком средстве контроля версий. Члены команды имеют доступ ко всем предыдущим версиям каждого варианта использования, для которых фиксируется история изменений. Аналитику требований этого проекта и его заместителю, отвечающему за архивирование, предоставлен доступ на чтение и запись; остальным членам команды — доступ только на чтение.

Наиболее надежный метод контроля версий заключается в хранении требований в базе данных коммерческого средства управления требованиями, как описано в главе 21. Эти средства отслеживают полную историю изменений требований, что особенно важно, если нужно вернуться к

более ранней версии требования. Такое средство позволяет вносить комментарии, где можно разумно обосновать решение о добавлении, изменении или удалении требования; эти комментарии могут оказаться полезными при необходимости обсуждения требования в будущем.

Атрибуты требований

Представьте себе каждое требование в виде объекта, который обладает свойствами, отличающими его от других требований. В дополнение к тестовому описанию, каждое функциональное требование должно иметь несколько поддерживающих его фрагментов информации или ***атрибутов*** (attributes), связанных с ним. Эти атрибуты представляют содержимое и подноготную каждого требования, они располагаются за описанием предполагаемой функциональности. Вы можете сохранить атрибуты в таблице, базе данных или, что наиболее эффективно, в средстве управления требованиями. Последние предоставляют несколько сгенерированных системой атрибутов, а также предоставляют возможность определить другие атрибуты. Эти средства позволяют фильтровать, сортировать выбранные подмножества требований на основании значений их атрибутов и запрашивать базу данных для их просмотра. Например, вы можете вызвать список всех высокоприоритетных требований, которые Шерп должна реализовать в версии 2.3 и которые имеют статус ***Approved*** (Одобрено).

Большой набор атрибутов особенно важен для громадных и сложных проектов. Возможно, вам следует указать для каждого требования такие атрибуты, как:

- дата создания требования;
- номер его текущей версии;
- автор требования;
- лицо, ответственное за удовлетворение требования;
- ответственный за требование или список заинтересованных лиц (чтобы принимать решения о предложенных изменениях);
- состояние требования;
- происхождение или источник требования;
- логическое обоснование требования;
- подсистема (или подсистемы), для которых предназначено требование;
- номер версии продукта, для которого предназначено требование;
- используемый метод проверки или критерий тестирования приемлемости;
- приоритет реализации;
- стабильность (показатель того, какова вероятность изменения требования в будущем; нестабильность требований может показывать плохо определенные или изменчивые бизнес-процессы или бизнес-правила).

Для чего же нужно это требование?

Менеджер продукта в компании, выпускающей электронные измерительные приборы, хотел просто записать включенные требования, потому что продукт конкурентов обладал теми же функциями. Такие функции хорошо отмечать с помощью атрибута ***Rationale*** (Логическое обоснование), позволяющего указать, почему конкретное требование включено в продукт.

Выберите наименьший набор атрибутов, с помощью которого вам удастся эффективно управлять проектом, Например, вряд ли вам одновременно потребуются и имя того, кто отвечает за требование, и название подсистемы, в которой расположено требование. Если какая-либо информация об атрибуте хранится в другом месте, например в общей системе контроля разработки, не следует копировать его в базе данных требований.

Ловушка

Множество атрибутов требований могут настолько озадачить разработчиков. что они просто не смогут предоставить все значения для всех атрибутов и эффективно использовать информацию об атрибутах. Начните с трех или четырех ключевых атрибутов. Добавьте другие атрибуты, когда вы будете знать, каким образом они улучшат ваш проект.

Основные версии требований динамичны. Набор требований, запланированный для определенной версии, будет изменяться по мере добавления новых положений и удаления или отсрочки до более поздних версий существующих требований. Разработчики могут жонглировать» отдельными документами требованиями для текущего проекта и будущих версий. Управление этими изменяющимися базовыми версиями и спецификацией требований на их основе — нудное дело. Если отложенные требований или те, от которых вы решили отказаться, остаются в спецификации требований, читатели спецификации могут запутаться в том, какие требования включены в конкретную базовую версию. Однако вам вряд ли захочется тратить много времени впустую, перетаскивая требования из одной спецификации требований в другую. Один из способов решения этой проблемы — сохранить требования в средстве управления требованиями и определить атрибут ***Release Number*** (Номер версии). Отсрочка требования означает изменение его запланированного выпуска, поэтому просто обновив номер версии, вы передвинете требование в другую базовую версию. Теми требованиями, от которых вы решили отказаться, лучше всего управлять с помощью атрибута статуса, как описано в следующем разделе.

На определение и обновление этих атрибутов требуется часть затрат, отведенных на управление требованиями, однако эти инвестиции в значительной степени окупаемы. В одной компании периодически генерировался отчет о требованиях, в котором показывалось, какое из 750 требований в трех связанных спецификациях назначено каждому проектировщику. Один из проектировщиков обнаружил несколько требований, которые он не считал своими. Он рассчитал, что сэкономил от одного до двух месяцев, отведенных на переделку проекта, которые бы понадобились, если бы он выяснил ситуацию с этими требованиями позднее.

Контроль статуса требований

«Как движется работа над той подсистемой, Джеки», — спросил Дейв.
«Довольно хорошо, Дейв. Готово около 90%».
Дейв был озадачен: «А разве пару недель назад ты не сделала примерно 90%?»
Джеки ответила: «Я думала, что да, но теперь эти 90% действительно готовы».

Иногда разработчики ПО слишком оптимистичны, когда сообщают об окончании задачи. Часто они выдают сами себе кредит, считая выполненными те задачи, к которым они приступили, но еще не закончила. Они могут сделать первоначально определенную работу на 90%, а затем столкнуться с непредвиденными трудностями. Тенденция переоценивать выполнение работы приводит к ситуации, типичной для всех проектов по разработке ПО или крупных задач, когда в течение долгого времени сообщается, что выполнено 90% объемов. Контроль статуса каждого функционального требования на протяжении всей разработки позволяет более точно оценивать готовность проекта. Ожидание же «выполнения задач» означает только повторы. Например, вы планируете реализовать в следующей версии только часть варианта использования, оставив полную реализацию до будущих версий. В этом случае вы будете отлеживать состояние тех функциональных требований, которые запланированы для ближайшей версии, поскольку этот набор требований должен быть выполнен на 100% до выпуска продукта.

Ловушка

Существует старая шутка, что на первую половину проекта по разработке тратится 90% ресурсов, а на остальную половину — оставшиеся 90% ресурсов. Чересчур оптимистичная оценка и попустительское отношение к контролю статуса верный путь к перерасходам в проекте.

В таблице 18-1 перечислено несколько возможных состояний требований. (Альтернативная схема показана в Caputo, 1998.) Некоторые специалисты добавляют состояние ***Designed*** (Разработано) (если элементы проектирования, в которых отражены функциональные требования, созданы и проверены) и ***Delivered*** (Выпущено) (ПО отдано в руки пользователей, как для **бета-тестирования).** Полезно отслеживать требования, от которых вы **отказались,** и причины этого, потому что отвергнутые требования имеют обыкновение «всплывать» в ходе разработки. Статус ***Rejected*** (Отклонено) позволяет оставить предложенное требование доступным для будущего использования, не создавая беспорядка в наборе утвержденных требований для определенного выпуска.

Распределение требований по нескольким категориям состояния имеет больший смысл, чем попытка контролировать процент завершения каждого требования или всей базовой версии. Определите, кто может изменить состояние требования, и обновляйте статус, только когда удовлетворены определенные условия перехода. Изменение состояния также ведет к обновлению данных о трассируемости требований, когда указывается, в каких элементах дизайна, кода и тестирования отражено требование (табл. **18-1).**

Таблица 18-1. Рекомендуемые состояния требования

Состояние	Определение
Proposed (Предложено)	Требование запрошено авторизированным источником
Approved (Одобрено)	Требование проанализировано, его влияние на проект просчитано, и оно было размещено в базовой версии определенной версии. Ключевые заинтересованные в проекте лица согласились с этим требованием, а разработчики ПО обязались реализовать его
Implemented (Реализовано)	Код, реализующий требование, разработан, написан и протестирован. Требование отслежено до соответствующих элементов дизайна и кода
Verified (Проверено)	Корректное функционирование реализованного требования подтверждено в соответствующем продукте. Требование отслежено до соответствующих вариантов тестирования. Теперь требование считается завершенным
Deleted (Удалено)	Утвержденное требование удалено из базовой версии. Опишите причины удаления и назовите того, кто принял это решение
Rejected (Отклонено)	Требование предложено, но не запланировано для реализации ни в одной будущих версий. Опишите причины отклонения требования и назовите того, кто принял это решение

На рис. 18-2 показано, как контролировать состояние требования в гипотетическом 10-месячном проекте: на графике показано, сколько процентов требований к системе в каждом состоянии имеется в кон не каждого месяца. Обратите внимание, что при этом не устанавливается, изменяется ли со временем количество требований в базовой версии. Кривые иллюстрируют, как достигается такая цель проекта, как полная проверка всех утвержденных требований. Основная часть работы считается выполненной, когда всем требованиям назначено состояние ***Verified*** (Проверено) (реализуется, как запланировано) или ***Deleted*** (Удалено) (удалено из базовой версии).

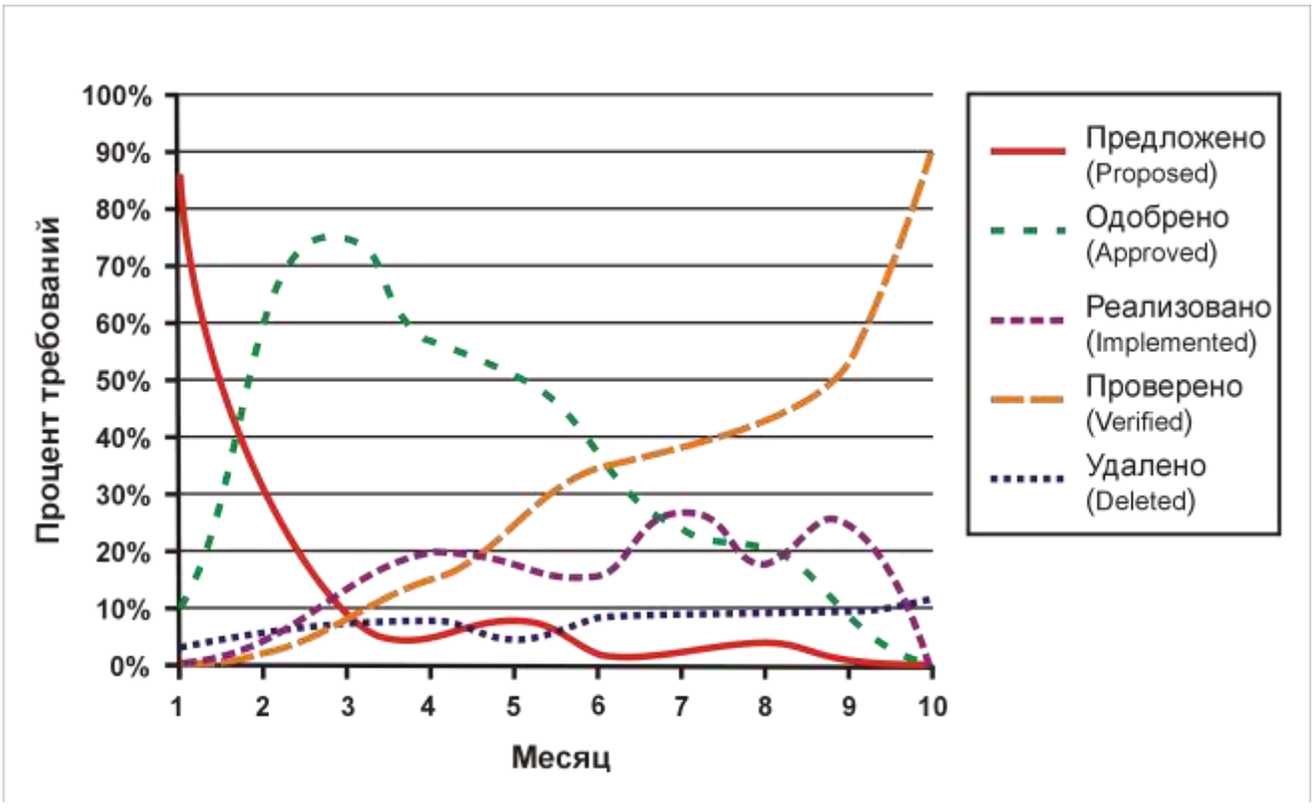


Рис. 18-2. Контроль состояния требований для цикла разработки проекта

Измерение усилий, необходимых для управления требованиями

Как и при разработке требований, в ваш план проекта должны быть включены задачи и ресурсы для выполнения задач по управлению требованиями, описанные в этой главе. Если вы выясните, сколько усилий тратится на управление требованиями, вы сможете оценить — мало их, как надо или слишком много — и изменить рабочие процессы и будущие планы соответствующим образом.

В организации, где никогда не измерялась ни одна из сторон работы, как правило, трудно контролировать, как члены команды тратят их рабочее время. Для измерения непосредственных усилий на разработку и управление проектом необходимо заняться корпоративной культурой и дисциплиной сотрудников — только так удастся фиксировать ежедневную работу. На это уходит не так много времени, как многие боятся, Команда сможет получить ценные идеи, поняв, как на самом деле тратятся усилия на различные задачи проекта (Wiegers, 1996a). Обратите внимание, что рабочие усилия — это не то же самое, что прошедшее календарное время. Выполнение задач может прерываться или возникает необходимость переговорить с другими людьми, что ведет к задержкам. Прилагаемые усилия, необходимые для выполнения задачи и измеряемые в человеко-часах, не обязательно изменятся из-за этих факторов, однако продолжительность выполнения задачи увеличится.

Контроль усилий также позволяет выяснить, выполняют ли разработчики предполагаемые задачи для управления требованиями. Неудача в управлении требований увеличивает риск появления неуправляемых изменений и требований, по неосторожности пропущенных в ходе реализации. Считайте усилия для выполнения следующих действий усилиями для управления требованиями:

- предложение изменения требований и новых требований;
- оценка предложенных изменений, включая оценку влияния изменения;
- изменение работы совета по управлению;
- обновление документации требований или базы данных;
- сообщение об изменениях требований заинтересованным группам и отдельным лицам;
- контроль и отчет о состоянии требования;
- сбор информации о трассируемости требований.

Управление требованиями помогает убедиться, что усилия, затраченные на сбор, анализ и документацию требований не потрачены впустую. Например, если вы не сможете своевременно обновлять набор требований в соответствии с вносимыми изменениями, то заинтересованным в проекте лицам будет трудно представить функциональность в каждой версии. Эффективное управление требованиями может уменьшить неверные ожидания: заинтересованные лица будут проинформированы о текущем состоянии требований в ходе процесса разработки.

Что теперь?

- Определите схему контроля версий для определения документации требований. Задокументируйте схему как часть процесса управления требованиями.
- Выберите состояния, которые вы хотите использовать для описания жизненного цикла функциональных требований. Нарисуйте диаграмму перехода состояний, чтобы показать условия, вызывающие изменения одного состояния на другое.
- Определите текущее состояние для каждого функционального требований в базовой версии. Обновляйте состояние по мере разработки.
- Опишите процессы, которым будет следовать ваша организация для управления требованиями в каждом проекте. Попросите нескольких аналитиков нарисовать, просмотреть, попробовать выполнить и одобрить действия и результаты. Этапы процесса, которые вы определите, должны быть практичными и выполнимыми, и они должны добавлять определенную ценность проекту.

Глава 19 Изменения случаются

«Как продвигается разработка, Гленн?» — спросил Дейв, менеджер проекта Chemical Tracking System на совещании, посвященному состоянию проекта.
«Я не так далеко продвинулся, как рассчитывал, — признался Гленн. — Сейчас я добавляю новую функцию запроса в каталоге для Шэрон, и это требует намного больше времени, чем я предполагал».
Дейв был озадачен. «Я не помню, чтобы мы обсуждали новую функцию запроса каталога на недавней встрече совета по управлению изменениями. Шэрон подала этот запрос через процесс контроля изменений?»
«Нет, она обратилась ко мне напрямую с этим предложением, — сказал Гленн. — Мне следовало попросить ее оформить его, как официальный запрос на изменение, но все выглядело настолько просто, что я сказал ей, что поработаю над этим. Выяснилось, что не все так просто. Каждый раз, когда мне кажется, что закончил, я понимаю, что я пропустил изменение, которое должно быть в другом файле, поэтому мне приходится исправлять это, собирать компонент и заново тестировать его. Я думал, мне хватит шести часов, однако я уже потратил на это четыре дня. Вот почему я не закончил другие, запланированные, задачи. Знаю, что задерживаю следующую сборку. Мне закончить с этой функцией или заняться тем, над чем я работал раньше?»

Большинство разработчиков сталкивались с простыми на первый взгляд изменениями, которые оказывались более сложными, чем они ожидали. Разработчики иногда не выполняют — или не могут выполнить — реалистичные расчеты затрат и других последствий предложенного изменения ПО. И когда разработчик, который хочет быть любезным, соглашается добавить улучшение, запрашиваемое пользователем, требования изменяются через «заднюю дверь», а не утверждаются заинтересованными лицами, которые имеют на это право. Такие неконтролируемые изменения — повсеместный источник хаоса в проекте, нарушения графика и проблем с качеством, особенно если работа ведется на нескольких участках или проект выполняет сторонняя организация. Организация, серьезно относящаяся к управлению своими проектами по разработке ПО, должна убедиться, что:

- предложенные изменения требований тщательно оценены до их ввода в дело;
- соответствующие лица принимают бизнес-решения, будучи хорошо информированными о запрошенных изменениях; одобренные изменения доведены до сведения всех участников проекта;
- изменения требований, внесенные в проект, согласованы друг с другом.

Изменение ПО само по себе неплохое дело. Виртуально определить все требования к продукту практически невозможно, а по мере прогресса изменяются и сопутствующие обстоятельства. Эффективная команда разработчиков ПО быстро отреагирует на необходимые изменения, чтобы продукт, который они создают, был своевременно поставлен клиентам.

Однако у изменений всегда есть цена. Простая Web-страница корректируется быстро и легко; новшества же в проекте интегрированной микросхемы могут обойтись в десятки тысяч долларов. Даже на отклонение запроса на изменение тратятся ресурсы, необходимые для подачи, оценки и принятия решения. Если заинтересованные в проекте лица не управляют изменениями в ходе разработки, им вряд ли известно, что именно будет получено в результате, а это неизбежно ведет к расхождениям в ожиданиях. Чем ближе выпуск версии, тем больше вам следует противиться желанию внести изменения в эту версию, поскольку последствия изменений становятся все более серьезными.

Обязанность аналитика требований — включить утвержденные изменения в проектную документацию требований. Принцип все тот же: документация требований должна точно описывать поставляемый продукт. Если вы не будете обновлять спецификацию требований по мере разработки продукта, его ценность будет снижаться, и команде придется работать таким образом, как будто бы у нее вообще нет никакой спецификации.

Если вам нужно внести изменение, начните с наивысшего уровня абстракции, затрагиваемого этим изменением, и далее влияйте на остальные компоненты системы через связанные с ним системные компоненты. Например, предложенное изменение может повлиять на вариант использования и его функциональные требования, однако никак не затронет ни одно бизнес-требование. Изменение требования к системе высокого уровня может воздействовать на множество требований к ПО. Некоторые изменения касаются только внутренних компонентов системы, так, например, реализуется уровень взаимодействия. Они невидимы для пользователя и скорее относятся к изменениям проекта или кода.

Если разработчик реализует изменение требования напрямую в коде, не пропуская его через описание требований и проектирование, возможны проблемы. Описание функций продукта, как оно приведено в спецификации требований, становится менее точным, поскольку код — это конечная реальность ПО. Код может стать неустойчивым и хрупким, если изменения вносятся без соотношения с архитектурой и структурой проекта. В одном проекте разработчики ввели новую и изменили существующую функциональность, о чем остальные члены команды не знали до тестирования системы. При этом понадобилась незапланированная переработка процедур тестирования и пользовательской документации. Последовательные приемы контроля изменений помогают предотвратить такие проблемы, а также связанные с этим расстройства, переделки и трату времени на тестирование.

Управление незапланированным ростом объема

Capers Jones (1994) сообщает, что незапланированный рост объема ставит под удар:

- 80% проектов по разработке систем управления информацией;
- 70% проектов по разработке военных систем ПО;
- 45% проектов по созданию ПО, выполняемых по контракту.

Незапланированным изменением требований считается предложение новой функциональности и существенной модификации после утверждения базовой версии требований к проекту. Чем дольше продолжается работа над проектами, тем больше их объем. Требования к системе управления информацией, как правило, увеличиваются на 1% в месяц (Jones, 1996b). Темп прироста может составлять до 3,5% в месяц для коммерческого ПО, при этом темпы роста других типов проектов входят в этот показатель. Проблема заключается не в изменении требований, а в том, что запоздалые изменения значительно влияют на уже выполненную работу. Если каждое предложенное изменение одобряется, то тем кто финансирует проект, его участникам и клиентам, может казаться, что проект никогда не будет завершен — и действительно такая возможность существует,

Определенные изменения требований абсолютно приемлемы, неизбежны и даже благоприятны. Бизнес-процессы, рыночные возможности, конкурирующие продукты, и технологии — все они могут меняться в ходе разработки продукта, и менеджеры могут определить, как в ответ на эти изменения необходимо откорректировать направление работы. Неуправляемый рост объема, при котором в проект постепенно включается новая функциональность без учета ресурсов, графика или целей качества, гораздо коварнее. Небольшая модификация здесь, неожидаемое улучшение там, и скоро нет никакой надежды разработать продукт соответствующего качества, который клиенты ожидают к определенному сроку.

Первый шаг при управлении незапланированным ростом объема — документирование образа, границ и ограничений новой системы, как части бизнес-требований (см. главу 5). Оценивайте каждое предложенное требование или функцию, соотносясь с бизнес-целями, образом продукта и границами проекта. Если задействовать клиентов в сборе информации, то снижается количество требований, упущенных из внимания, которые приходится добавлять к задачам команды после того, как обязательства приняты, а ресурсы распределены (Jones, 1996a). Составление прототипов - это еще один эффективный прием управления незапланированным ростом объема (Jones, 1994). Прототип позволяет предварительно реализовать продукт, что помогает разработчикам и пользователям прийти к общему пониманию нужд пользователей и необходимых решений. Короткие циклы разработки при инкрементальной разработке системы позволяют вносить изменение часто, это особенно полезно, когда требования неясные или быстро изменяются (Beck, 2000).

Наиболее эффективный прием для управления незапланированным ростом объема - это умение сказать: «Нет» (Weinberg, 1995). Люди в принципе не любят отказывать, а разработчиков могут

прессинговать по поводу каждого предложенного требования. Такие принципы, как «Клиент всегда прав» и «Мы сделаем все, чтобы клиент остался доволен» звучат прекрасно в теории, однако за их выполнение нужно платить. Игнорируя это, вы никак не измените тот факт, что коррективы не могут произойти просто так. Президент одной компании, специализирующейся на поставке программного обеспечения, привык слышать от менеджера по разработке в ответ на предложение новой функции: «Не сейчас». «Не сейчас», — звучит более привлекательно, чем простой отказ. В этом слышится обещание включить функцию в следующие версии. Включение каждой функции, запрашиваемой клиентами, специалистами отдела маркетинга, менеджерами продукта и разработчиками, приводит к размыванию обязательств, снижению качества, истощению разработчиков и созданию ПО с избыточной функциональностью. Клиенты не всегда правы, но в их высказываниях всегда есть смысл, поэтому фиксируйте их идеи — возможно, их стоит включить в последующие версии.

В идеальном мире вам удастся собрать все требования к новой системе до начала сборки, и они останутся стабильными в течение всей разработки. Это основа последовательной или водопадной модели разработки ПО, но на практике так не бывает. На определенном этапе вам необходимо «заморозить» требования до определенной версии или вам никогда не удастся получить результат. Однако, если вы приостановите работу над требованиями слишком рано, то вы проигнорируете то обстоятельство, что клиенты не всегда уверены в том, что именно им нужно; клиентам понадобились изменения, и разработчикам необходимо отреагировать на это. Для каждого проекта следует предусмотреть включение определенных изменений в проект, но контролируемым способом.

Ловушка

«Заморозить» требования для новой системы после выполнения первоначального сбора информации - не умно и не практично. Лучше определите базовую версию, когда решите, что требования достаточно хорошо определены, чтобы начать дизайн и сборку, а затем управляйте неизбежными изменениями, чтобы минимизировать их негативное влияние на проект.

Процесс контроля изменений

Как-то, оценивая процесс разработки ПО, я спросил специалистов, работающих над проектом, как они включают изменения в требования к продукту. После неловкой паузы, один из них сказал: «Каждый раз, когда специалист отдела маркетинга хочет внести изменение, он обращается к Брюсу или Сэнди, потому что они всегда соглашаются, а мы нет». Эта процедура внесения изменений не показалось мне хорошей.

Процесс контроля изменений позволяет руководителю проекта принимать информированное бизнес-решение, которое удовлетворяет клиента и имеет коммерческую ценность, при этом контролируются затраты на жизненный цикл продукта. Вы можете отслеживать статус всех предложенных изменений и убедиться, что предложенные требования не были потеряны или упущены. После утверждения базовой версии набора требований придерживайтесь этого порядка для внесения всех предлагаемых изменений в нее.

Клиенты и другие заинтересованные лица часто уклоняются от нового процесса, однако процесс контроля изменений — не препятствие для модификации. Это механизм суммирования и фильтрации, дающий уверенность, что в проект включены наиболее ценные изменении. Если предложенное изменение не настолько важно, чтобы заинтересованное в проекте лицо потратило пару минут на передачу его через стандартные, простые каналы, то стоит задуматься о его ценности вообще. Процесс управления должен быть тщательно задокументирован, максимально прост и, что важнее всего, эффективен.

Ловушка

Если вы предложите заинтересованным лицам новый процесс контроля изменений - неэффективный, громоздкий или слишком сложный, они постараются найти способ избежать этого - и, возможно, будут правы.

Контроль изменений требований схоже с процессом сбора и принятия решений по отчетам об ошибках. Эти же средства могут поддерживать обе задачи. Однако следует помнить — инструмент не заменяет процесс. Использование коммерческого средства отслеживания проблем для управления предложенными изменениями требований не заменит описание процесса, где объясняются содержимое и обработка запроса на изменение.

Политика контроля изменений

Руководство должно ясно довести до сведения сотрудников политику, в которой описываются ожидания того, как участники проекта должны обрабатывать предложенные изменения требований. Политики имеют смысл, только если они реалистичны, добавляют ценность и проводятся в жизнь. Я обнаружил, что могут быть полезными следующие положения политики контроля изменений:

- все изменения требований должны вноситься в соответствии с процессом. Если запрос на изменение подается каким-то иным способом, он не рассматривается;
- нельзя заниматься проектированием или реализовать неутвержденные изменения, кроме проверки их осуществимости;
- сам по себе запрос на изменение не гарантирует выполнение этого изменения. ***Совет по управлению изменениями проекта*** (change control board, CCB) решает, какие изменения следует реализовать. (Мы поговорим о совете по управлению изменениями проекта далее в этой главе.);
- содержимое базы данных изменений должно быть видимым для всех заинтересованных в проекте лиц;
- первоначальный текст запроса на изменение не должен изменяться или удаляться;
- анализ влияния изменения необходимо выполнять для каждого изменения;
- каждое включенное изменение требования должно отслеживаться вплоть до утверждения запроса на это изменение; обоснование каждого утверждения или отклонения запроса на изменение необходимо документировать.

Разумеется, мелкие изменения вряд ли повлияют на проект, а существенные воздействуют довольно сильно. В принципе, любые из них следует обрабатывать с помощью процесса контроля изменений. На практике вы можете делегировать принятие определенных детальных решений, касающихся требований, разработчикам, однако ни одно решение, влияющее на работу более чем одного человека, не должно идти в обход процесса контроля изменений. Однако процесс должен предлагать «быстрый путь», чтобы ускорить не рискованные и дешевые изменения в сжатом цикле принятия решений.

Ловушка

Разработчики не должны использовать процесс контроля изменений как барьер для задержки выполнения каких-либо изменений. Если изменение неизбежно — работайте над ним.

Описание процесса контроля изменений

На рис. 19-1 показан шаблон описания процесса контроля изменений для обработки модификаций требований и других изменений проекта. Вы можете загрузить его образец с *http://www.processimpact.com/goodies.shtml*. Далее речь пойдет в основном о том, как процесс будет

обрабатывать изменения требований. Я считаю полезным включить следующие четыре компонента во все процедуры и описания процесса:

- входной критерий — условия, которые должны быть удовлетворены до выполнения процесса или процедуры;
- различные задачи, задействованные в процессе или процедуре;
- участник проекта, отвечающий за выполнение каждой задачи, и другие лица, принимающие участие в выполнении задачи;
- последовательные действия для подтверждения того, что задачи были выполнены правильно;
- выходной критерий — условия, указывающие, когда процесс или процедура считаются успешно завершенными.

1. Введение 1.1. Назначение 1.2. Границы 1.3. Определения
2. Роли и ответственности
3. Состояние запроса на изменение
4. Входной критерий
5. Задачи 5.1. Оценка запроса 5.2. Принятие решения 5.3. Внесение изменения
6. Проверка 6.1. Проверка изменения
7. Выходной критерий
8. Отчет о состоянии контроля изменений
Приложение. Элементы данных, сохраненные для каждого запроса

Рис. 19-1. Пример шаблона описания процесса контроля изменений

1.Введение

Во введении описывается назначение процесса и определяются организационные границы, в которых он применяется. Если этот процесс затрагивает только изменения в определенных продуктах, их следует определить здесь. Также укажите, существуют ли какие-то специальные виды изменений, которые не подлежат контролю, например изменения в промежуточных или служебных продуктах, созданных в ходе проекта. В разделе 1.3 определите все термины, необходимые для понимания остальной информации.

2. Роли и ответственности

Перечислите членов команды — по ролям, не по именам, которые участвуют в процессе контроля изменений, и опишите их обязанности. В табл. 19-1 приведены некоторые типичные роли; адаптируйте их в соответствии с конкретной ситуацией. Каждый человек может выполнять несколько ролей. Например, председатель совета по управлению изменениями также может получать подаваемые запросы на изменения. В небольших проектах несколько, а возможно, и все роли выполняются одним человеком.

Таблица 19-1. Возможные роли участников проекта при управлении изменениями

Роль	Описание и ответственность

Председатель совета, по управлению изменениями	Возглавляет совет по управлению изменениями; как правило, обладает правом принятия окончательного решения, если совет не приходит к согласию; выбирает тех, кто оценивает и вносит изменения для каждого запроса на изменение.
Совет по управлению изменениями	Группа, принимающее решение утвердить или отклонить каждое предложенное изменение для определенного проекта.
Тот, кто оценивает изменение	Человек, которого председатель совета просит проанализировать влияние предложенного изменения; это может быть сотрудник технического отдела, клиент, маркетолог или сотрудник, занимающийся всем этим понемногу.
Тот, кто вносит изменение	Отвечает за внесение изменений в продукт, когда запрос: на изменение утвержден.
Тот, кто инициирует запрос	Некто, подающий новый запрос на изменение.
Получатель запроса	Лицо, которому передаются новые запросы на изменение.
Тот, кто проверяет изменение	Человек, определяющий, корректно ли выполнено изменение.

3. Состояние запроса на изменение

Запрос на изменение проходит через определенные стадии на протяжении жизненного цикла, причем на каждой стадии он имеет различные статусы. Вы можете представить эти изменения состояния, используя диаграмму перехода состояний, как показано на рис. 19-2. Обновляйте состояние запроса, только когда удовлетворяются определенные критерии.

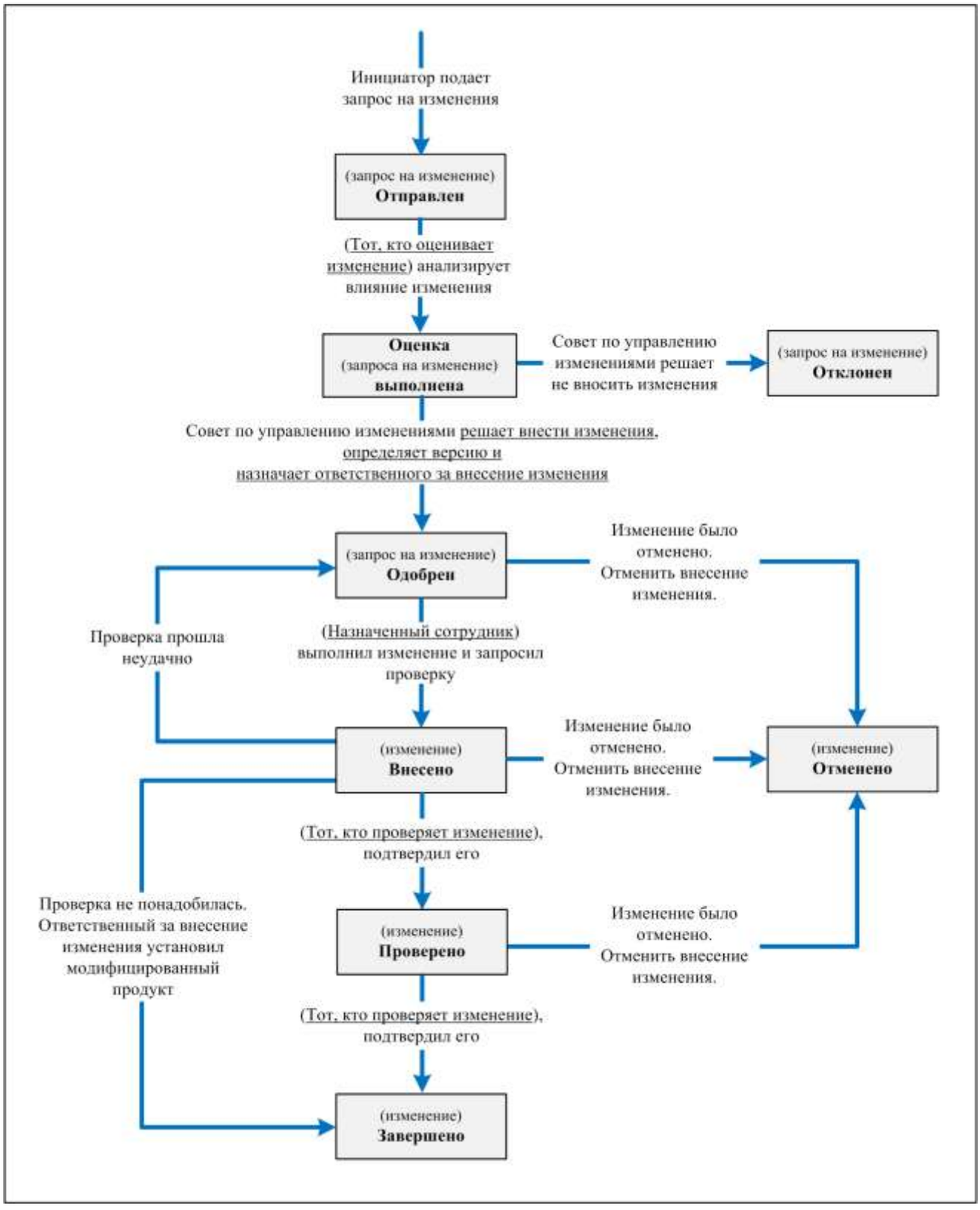


Рис. 19-2.Диаграмма перехода состояний для запроса на изменение

4. Входной критерий

Основным входным критерием для процесса контроля изменений является действительный запрос на изменение, полученный по утвержденным каналам.

Все потенциальные инициаторы запросов должны знать, как подавать запросы на изменение, независимо от того, отправляют они его, заполняя бумажную форму или Web-форму, отправляя сообщение по электронной почте или используя средство контроля изменений. Назначьте каждому запросу на изменение уникальный идентификатор и направьте их всех к единой точке контакта —

получателю запроса.

5. Задачи

Следующий шаг — это оценка запроса на предмет технической выполнимости, затрат и соответствия бизнес-требованиям проекта и ограничениям ресурсов. Председатель совета по управлению изменениями может назначить сотрудника для оценки влияния изменения, анализа риска, анализа опасности и др. Анализ позволяет убедиться, что вероятные последствия изменения ясны. Тот, кому поручена оценка, и члены совета по управлению изменениями должны также рассмотреть коммерческие и технические последствия отклонения изменения.

Затем уполномоченные члены совета по управлению изменениями решают, утвердить или отклонить запрошенное изменение. Совет по управлению изменениями присваивает каждому одобренному изменению уровень приоритета, или назначает дату реализации, или назначает это изменение определенной сборке или номеру версии. Далее совет по управлению изменениями оповещает о решении, обновляя состояние запроса или уведомляя всех членов команды, которые занимаются модификацией. После этого необходимо обновить документацию требований, описание проекта и моделей, компоненты пользовательского интерфейса, код, документацию тестирования, экраны справки и руководства для пользователей. Тот, кто вносит изменения, делает это установленным путем.

6. Проверка

Изменения требования, как правило, проверяют с помощью экспертной оценки, чтобы убедиться, что измененные требования, варианты использования и модели соответствующим образом отражают все аспекты изменения. Информация о трассируемости поможет вам найти все части системы, которые это изменение затрагивает и которые, как следствие, необходимо проверить. Поручите нескольким членам команды проверить изменения с помощью тестирования или экспертизы. После этих процедур тот, кто занимается проверкой, устанавливает обновленные продукты, сделав их доступными остальным членам команды, и переопределяет базовую версию, в которой учтены изменения.

7. Выходной критерий

Все перечисленные далее выходные критерии должны быть удовлетворены, чтобы должным образом завершить процесс управлений изменениями:

- состояние запроса: ***Rejected*** (Отклонено), ***Closed*** (Закрыто) или ***Canceled*** (Отменено); все изменения записаны в соответствующих разделах;
- инициатор запроса, председатель совета по управлению изменениями, менеджер проекта и другие значимые участники проекта оповещены о деталях изменения и текущем состоянии запроса на изменение;
- матрица связей требований обновлена. (В главе 20 о трассируемости требований рассказано более подробно.)

8. Отчет о состоянии управления изменения

Определите графики и отчеты, которые вы будет использовать при обобщении содержимого базы данных контроля изменений. Обычно на этих графиках показано количество запросов на изменение в каждой категории состояния как функция времени. Опишите процедуры создания графиков и отчетов. Менеджер проекта использует эти отчеты при контроле состояний проекта.

Приложение. Элементы данных, сохраненные для каждого запроса

В табл. 19-2 перечислены некоторые элементы данных, которые можно сохранить для каждого запросе на изменение. После определения своего собственного списка, укажите, какие элементы следует сохранять обязательно, а какие нет. Также укажите, устанавливается ли значение для каждого элемента автоматически средством контроля изменений или вручную одним из участников процесса управления изменениями.

Таблица 19-2. Предлагаемые элементы данных запроса на изменение

Элемент	Описание
Происхождение изменения	Функциональная область, запросившая это изменение; это могут быть маркетологи, менеджеры, клиенты, инженеры по программному обеспечению, специалисты по оборудованию и тестировщики
Идентификатор запроса на изменение	Идентификатор или порядковый номер, назначенный запросу
Тип изменения	Тип запроса на изменение, например изменение требования, предложенное улучшение или отчет об ошибке
Дата подачи	Дата, когда инициатор запроса отправил запрос на изменение
Дата обновления	Дата последнего изменения запроса
Описание	Текстовое описание в свободной форме запрошенного изменения
Приоритет при реализации	Относительная важность внесения изменения, определенная советом по управлению изменениями - низкая, средняя или высокая
Лицо, которое вносит изменения	Имя человека, отвечающего за внесение изменения
Инициатор запроса	Имя человека, подавшего запрос на изменение; возможно, вам следует включить контактную информацию о нем
Приоритет инициатора запроса	Относительная важность изменения с точки зрения инициатора запроса - низка, средняя или высокая
Планируемый выпуск	Версия продукта или номер сборки, на которую запланировано одобренное изменение
Проект	Название проекта, для которого предлагается изменение
Ответ	Ответ в свободной текстовой форме на запрос об изменении; со временем их может быть несколько; не изменяйте существующие ответы при вводе нового
Состояние	Текущий статус запроса на изменение, выбранное из возможных на рис. 19-2
Название	Краткое (одна строчка) описание предложенного изменение
Лицо, проверяющее изменение	Имя человека, отвечающего за корректность изменения

Совет по управлению изменениями

Совет по управлению изменениями (change control board, CCB) (иногда его называют советом по управлению конфигурацией) был признан лучшим практическим решением при разработке ПО (McConnell, 1996). Это группа людей, независимо оттого, сколько их — один человек или несколько, принимающая решение о том, какие предложенные изменения требований и новые функции принять для включения в продукт. Совет по управлению изменениями также решает, какие выявленные ошибки следует исправить и когда. Многие проекты уже де-факто имеют такие группы; официальное назначение совета по управлению изменениями позволяет определить его структуру и полномочия, а также назначить рабочие процедуры.

Совет по управлению изменениями просматривает и утверждает изменения базовых версий любой документации проекта, например спецификации требований. Некоторые советы имеют полномочия для принятия решений и просто информируют менеджеров о внесении этих изменений, тогда как другие могут только давать рекомендации менеджерам для принятия решений. В небольшом проекте имеет смысл делегировать принятие решений одному или двум сотрудникам. В крупных проектах или программах может быть несколько уровней советов контроля изменений: одни отвечают за принятие бизнес-решений, например об изменении требований, а другие — за решения технического характера (Sorensen, 1999). Совет по управлению изменениями более высокого уровня утверждает изменения, очень сильно влияющие на проект. Например, при разработке большой программы объединяющей несколько проектов, следует создать совет, принимающий решения на уровне программы, и отдельные советы для каждого проекта. Последние разрешают проблемы и рассматривают изменения, влияющие только на конкретный проект. Вопросы, касающиеся других проектов, и изменения, в результате которых возможно превышение затрат или нарушение графика, передаются в совет уровня программы.

Для некоторых людей термин «совет по управлению изменениями» означает неэкономичные бюрократические накладные расходы. Однако это ценная структура, помогающая управлять даже небольшим проектом. Она не должна отнимать много времени или быть громоздкой — она должна работать эффективно. Это означает, что совет по управлению изменениями рассматривает все предложенные изменения быстро и принимает своевременные решения на основании анализа возможного влияния и преимуществ каждого предложения. Эта структура должна быть не больше и не официальнее, чем необходимо для того, чтобы удостовериться, что соответствующие лица принимают правильные бизнес-решения о каждом запрашиваемом изменении.

Состав совета по управлению изменениями

Члены совета по управлению изменениями должны представлять все группы, которым необходимо принимать участие в принятии решений в рамках полномочий совета. Рассмотрите участие представителей следующих областей:

- менеджеров проекта или программы;
- менеджеров продукта или аналитики требований;
- разработчиков;
- специалистов по тестированию или проверке качества:
- маркетологов или представителей клиента;
- специалистов, ответственных за пользовательскую документацию;
- специалистов технической службы или службы поддержки;
- специалистов по управлению конфигурацией.

Только некоторые из них будут принимать решения, однако всех их следует проинформировать о решениях, влияющих на их работу.

Такая же небольшая группа будет выполнять некоторые из этих ролей в небольших проектах, а тем, кто выполняет другие роли, не придется рассматривать каждый запрос на изменение. Если проект распространяется и на ПО, и на оборудование, в совет по управлению изменениями можно ввести специалистов по оборудованию, системных инженеров, специалистов по производству и, возможно, специалистов по проверке качества оборудования и управлению конфигурацией. Совет должен быть небольшим, чтобы удавалось быстро и эффективно реагировать на запросы на

изменение. Как большинство из вас уже поняли, большим группам трудно планировать проведение встреч и приходить к общему решению. Убедитесь, что члены совета понимают свои обязанности и относятся к ним серьезно. Для того чтобы убедиться, что совет владеет необходимой технической и бизнес-информацией, пригласите специалистов на совещание, где будут обсуждаться предложения, относящиеся к сфере деятельности этих специалистов.

Устав совета по управлению изменениями

В уставе описываются задачи, область полномочий, членство, производственные процедуры и процесс принятия решений совета по управлению изменениями (Sorensen, 1999). Шаблон устава совета доступен на *http://www.processimpact.com/goodies.shtml*. В уставе должна быть указана регулярность запланированных совещаний и условия созыва специальных встреч. Область полномочий указывает, какие решения совет может принимать, а какие следует передать совету более высокого уровня или менеджеру.

Принятие решений

Как и все структуры, принимающие решения, каждый совет по управлению изменениями должен определить соответствующие правила и процесс принятия решения (Gottesdiener, 2002). В описании процесса принятия решения следует указать следующее:

- сколько членов совета по управлению изменениями или лиц, выполняющих ключевые роли, составляют кворум для принятия решений;
- используется ли голосование, консенсус, консультативный способ или какое-либо другое правило принятия решений;
- может ли председатель совета по управлению изменениями отклонить коллективное решение совета;
- должен ли совет по управлению изменениями более высокого уровня или кто-либо еще ратифицировать решение.

Совет по управлению изменениями взвешивает предполагаемые преимущества и предполагаемое влияние принятия предложенного изменения. К преимуществам относятся: экономия средств, увеличение дохода, большее удовлетворение покупателей и конкурентные преимущества. Влияние от изменения учитывает негативный эффект, которое изменение может оказать на продукты или проект. К ним относятся: увеличение затрат на разработку и поддержку, задержка поставки, снижение качества продукта, уменьшение функциональности и недовольство клиентов. Если просчитанное влияние на затраты или график превышает допустимый порог полномочий для данного совета, передайте изменение для рассмотрения менеджменту или совету по управлению изменениями более высокого уровня. В противном случае процесс принятия решений совета по управлению изменениями поможет вам утвердить или отклонить предложенное изменение,

Уведомление о состоянии

После того как совет по управлению изменениями принимает решение, уполномоченный сотрудник обновляет состояние запроса в базе данных изменений. Некоторые средства автоматически генерируют сообщения электронной почты, чтобы информировать о новом состоянии инициатора запроса и всех, кого затрагивает это изменение. Если сообщение электронной почты не генерируется автоматически проинформируйте заинтересованных специалистов лично для того, чтобы они смогли должны образом принять это изменение.

Пересмотр обязательств

Было бы глупо предполагать, что заинтересованные лица могут включать все больше и больше функциональности в проект, который ограничен графиком, кадрами, бюджетом и качеством, и проект все равно останется успешным. До принятия важного изменения требования повторно обговорите обязательства с менеджерами и клиентами, чтобы принять это изменение (Humphrey, 1997). Вы можете просить увеличить сроки, привлечь больше специалистов или задержать незаконченные требования, обладающие низким приоритетом. Если вы не добьетесь определенного пересмотра обязательств, составьте список рисков проекта, ставящих под угрозу его успех, чтобы люди не удивлялись, если результаты не будут соответствовать желаемым.

Средства контроля изменений

Автоматизированные средства могут сделать ваш процесс контроля изменений более эффективным (Wiegers, 1996a). Многие команды используют коммерческие средства отслеживания проблем и вопросов для сбора, хранения и контроля изменений требований. Список предложений изменений, поданных за последнее время, сгенерированный с помощью такого средства, можете стать повесткой дня совещания совета по управлению изменениями. Средства отслеживания проблем также можно использовать для отчета о количестве запросов в каждой категории состояния в любой момент времени.

Поскольку доступные средства, поставщики и функции довольно часто меняются, я не буду рекомендовать конкретное средство. Обращайте внимание на следующие параметры средства для поддержки процесса изменения требований:

- позволяет ли определить элементы данных, включенные в запрос на изменение;
- позволяет ли определить модель перехода состояния для жизненного цикла запроса на изменение;
- приводит ли в действие модель перехода состояния, чтобы только авторизованные пользователи смогли вносить разрешенные изменения состояния; фиксирует ли дату изменения каждого состояния и лицо, выполнившее это изменение;
- позволяет ли определить, кто получил автоматическое уведомление по электронной почте при подаче нового запроса или обновлении состояния запроса;
- создает ли и стандартные, и нестандартные отчеты и графики для клиентов,

В некоторые коммерческие средства управления требованиями — они обсуждаются в главе 21 — встроена простая система изменения предложений. Эти системы могут связать предложенное изменение с определенным требованием, чтобы сотрудник, ответственное за каждое требование, получал уведомление по электронной почте при каждой подаче соответствующего запроса на изменение.

Оснащение процесса

Когда я работал в команде, занимающейся Web-разработками, одним из наших первых улучшений стала реализация контроля изменений для управления большим журналом запросов на изменение (Wiegers, 1999b). Мы начали с процесса, похожего на один из тех, что описан в этой главе. Мы пробовали его в течение нескольких недель, использую бумажные формы, пока я оценивал несколько программных средств отслеживания проблем. Мы нашли несколько способов улучшения нашего процесса и обнаружили несколько дополнительных элементов данных, которые требовались нам в запросах на изменение. Мы выбрали отлично конфигурируемое средство отслеживания проблем и настроили его с учетом специфики нашего процесса. Команда использовала этот процесс и средство для обработки изменений требований в системе, касающихся разработки, отчетов об ошибках и предложенных улучшений в производственных системах, обновлений содержимого Web-сайта и запросов, касающихся новых проектов по разработке. Контроль изменений стал одной из наших наиболее успешных инициатив по улучшению процесса.

Измерение активности изменений

Измерение ПО позволяет проникнуть в суть проекта, продуктов и процессов более точно, чем если опираться на субъективные впечатления или неясные воспоминания о прошлом опыте. При выборе показателей следует опираться на вопросы, на которые вы или менеджеры пытаетесь найти ответы, а также на цели, которых вы пытаетесь достичь. Измерение активности изменений — это

способ оценки стабильности требований. Он позволяет также обнаружить возможности для улучшения процесса, чтобы снизить количество изменений в будущем.

Возможно, вам следует контролировать следующие показатели активности изменений (Paulk и др., 1995):

- количество полученных запросов на изменение, открытых и закрытых в настоящее время;
- общее число пришедших запросов, включая добавленные, удаленные и измененные требования (вы также можете подсчитать этот показатель как процент от общего количества требований в базовой версии);
- количество полученных запросов на изменение, исходящих из каждого источника;
- количество изменений, предложенных и внесенных в каждое требование после создания базовой версии;
- общие усилия на обработку и реализацию запросов на изменение; количество циклов процесса изменений, необходимых для должной реализации каждого утвержденного изменения (иногда изменения реализуются неправильно или являются причиной появления других ошибок, которые необходимо исправить).

Вам не обязательно настолько тщательно отслеживать этапы изменения требований. Как и в случае с показателями ПО, необходимо до принятия решения о том, что именно вы будете измерять, четко понять цель измерения и то, как вы будете использовать полученные данные (Wiegers, 1999a). Начните с простых параметров, чтобы выработать культуру измерения в вашей организации и собрать ключевые данные, необходимые для эффективного управления проектами. По мере наработки опыта усложняйте ваши измерения, насколько это необходимо для успеха проекта.

На рис. 19-3 показан один из способов контроля количества изменений требований в ходе разработки проекта. Здесь отслеживается скорость появления предложения об изменении требований. Точно так же контролируется и количество утвержденных запросов на изменение. Не подсчитывайте изменения, внесенные до составления базовой версии; вы знаете, что в этот период требования все еще развиваются. Однако после создания базовой версии все изменения должны приниматься в соответствии с процессом контроля изменений. Вам также следует проследить частоту изменений (изменчивость требований). Кривая на этом графике должна стремиться к нулю по мере приближения даты поставки. Высокая частота в течение длительного времени свидетельствует о том, что возможно нарушение расписания поставки продукта. Вероятна и другая причина: первоначальная базовая версия требований была неполной, то есть следует улучшить подход к сбору информации по требованиям.

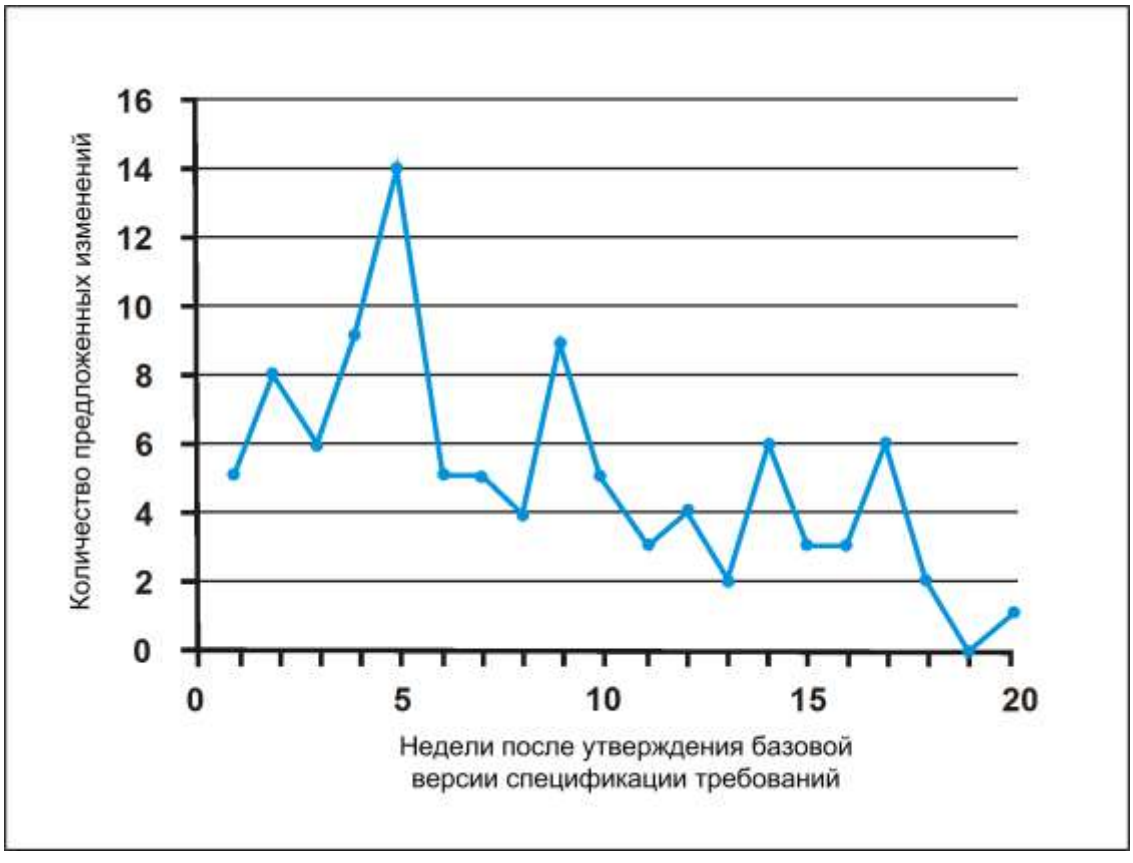


Рис. 19-3. Пример графика активности изменения требований

Рис. 19-4. Пример графика, демонстрирующего источники изменения требований

Менеджер проекта, обеспокоенный тем, что частые изменения помешают своевременному завершению проекта, может собрать дополнительную информацию, отследив происхождение изменения требований. На рис. 19-4 показан способ представления количества запросов на изменение, пришедших из разных источников. Менеджеру проекта стоит обсудить такой график с менеджером по маркетингу, так как большинство изменений запрошено именно отделом маркетинга. Плодотворная дискуссия позволит установить, какие действия должна предпринять команда, чтобы уменьшить число изменений, полученных после создания базовой версии из отдела маркетинга. Использование данных в качестве отправной точки для такой дискуссии гораздо продуктивнее

встречи противоборствующих сторон, основанной на взаимном раздражении.

Изменение не бесплатно: анализ влияния

Необходимость анализа результатов изменения при внесении значительных изменений очевидна. Однако неожиданные сложности могут вызывать самые незначительные коррективы, Как-то одной компании понадобилось изменить в продукте текст одного сообщения об ошибке. Казалось, что может быть проще? Существовали две версии продукта — английская и немецкая. В английской версии все было в порядке, но в немецкой размер сообщения превышал максимально допустимую длину сообщения об ошибке и в окне сообщения и в базе данных. Выполнение этого на первый взгляд простого задания оказалось более трудоемким, чем предполагал разработчик, когда обещал быстрое решение.

Ловушка

Поскольку люди не любят говорить «нет», то легко собрать огромный журнал одобренных запросов на изменение. Прежде чем принять изменение, убедитесь, что вы понимаете его логическое обоснование, его соответствие образу продукта и коммерческую ценность.

Анализ влияния — это ключевой аспект ответственного управления требованиями (Arnold и Bohner, 1996). Он обеспечивает точное понимание подтекста предложенного изменения, что помогает команде .принимать информированные бизнес-решения о том, какое изменение одобрить. Анализ позволяет выявить компоненты, которые может понадобиться создать, изменить или отклонить и оценить затраты, связанные с реализацией изменения. До того, как разработчик ответит: «Конечно, без проблем», он или она должны потратить время на анализ результата изменения.

Процедура анализа влияния изменения

Председатель совета по управлению изменениями обычно просит опытного разработчика выполнить анализ результата определенного. Анализа результатов изменений затрагивает три аспекта.

1.Определите возможны последствия изменения. Часто они вызывают значительный волновой эффект. Включение множества функций в продукт может снизить его производительность до неприемлемого уровня, например, когда системе, запускаемой ежедневно, потребуется 24 часа для завершения одного запуска.

2.Определите все файлы, модели и документы, которые, возможно, придется изменить, если команда включит все запрошенные изменения.

3.Определите задачи, необходимые для реализации изменения, и оцените усилия, необходимые для выполнения этих задач.

Ловушка

Пропустив выполнение анализа влияния, вы не измените объем задачи. Просто в этом случае этот объем станет сюрпризом для вас. А сюрпризы в области ПО редко бывают приятными.

На рис. 19-5 показан контрольный список вопросов, разработанный в помощь аналитику. В списке на рис. 19-6 перечислены наводящие вопросы, которые помогут вам определить все элементы ПО, затронутые изменением. Данные трассируемости, которые связывают изменяемые требования с другими основными документами проекта, крайне полезны при анализе влияния. По мере наработки опыта при использовании этих контрольных вопросов адаптируйте их для ваших проектов.

Далее описана несложная процедура оценки влияния изменения предложенного требования. Множество проблем с оценкой возникает так как специалист, выполняющий оценку, не принимает во внимание всю работу, необходимую для завершения задачи. Следовательно, этот подход к

анализу влияния требует тщательного определения задачи. Для оценки значительных изменений используйте небольшие команды — а не просто одного разработчика: так вам удастся не упустить из виду важные задачи.

1.Работайте со списком на рис. 19-5.

2.Работайте со списком на рис. 19-6, используя доступную информацию трассируемости. В некоторые средства управления требованиями входят отчеты об анализе влияния, в которых размещены ссылки трассируемости, следуя по ним удается найти элементы системы, зависящие от изменяемых требований.

■	Конфликтуют ли какие-то из требований в базовой версии с предложенным изменением?
■	Конфликтуют ли какие-то из отложенных требований в базовой версии с предложенным изменением?
■	Какие могут быть технические или бизнес-последствия, если изменение не будет внесено?
■	Какие могут быть побочные негативные эффекты или другие риски, если изменение не будет внесено?
■	Повлияет ли негативно предложенное изменение на требования к производительности и другие атрибуты качества?
■	Выполнимо ли предложенное изменение в рамках известных технических ограничений и квалификации специалистов?
■	Не потребуется ли для реализации предложенного изменения неприемлемое количество компьютерных ресурсов, необходимых для разработки, тестирования или операционной среды?
■	Нужно ли приобрести какие-либо средства для реализации и тестирования этого изменения?
■	Как предложенное изменение повлияет на последовательность, зависимости, усилия или продолжительность задач, включенных в настоящее время в план проекта?
■	Потребуется ли создание прототипов или другая информация пользователей для проверки предложенного изменения?
■	Сколько затрат, уже вложенных в проект, будут потеряны, если принять это изменение?
■	Не увеличится ли затрата на единицу продукта при принятии изменения, например за счет увеличения оплаты лицензирования продукта приглашенными специалистами?
■	Повлияет ли изменение на планы по маркетингу, производству, обучению или поддержки покупателей?

Рис. 19-5. Список возможных последствий предложенного изменения

■	Определите все необходимые изменения пользовательского интерфейса, дополнения или удаления.
■	Определите все изменения, добавления или удаления, которые необходимо внести в отчеты, базы данных или файлы.
■	Определите компоненты дизайна, которые придется создать, изменить или удалить.
■	Определите файлы исходного кода, которые необходимо создать, изменить или удалить.
■	Определите все изменения, которые придется внести в уже созданные файлы или процедуры.
■	Определите существующие варианты тестирования элементов продукта, целостности, системы и приемлемости, которые необходимо изменить или удалить.
■	Оцените необходимое количество новых вариантов тестирования элементов продукта, целостности, системы и приемлемости.
■	Определите все экраны справки, обучающие материалы или другую пользовательскую документацию, которую необходимо создать или изменить.
■	Определите все компоненты приложений, библиотек или оборудования, на которые повлияет изменение.
■	Определите все стороннее ПО, которое должно быть приобретено или лицензировано.
■	Определите влияние, которое предложенное изменение окажет на планы управления проектом ПО, проверки приемлемости качества, управления конфигурацией и другие планы.

Рис. 19-6. Список возможных элементов ПО, затрагиваемых изменением

3.Воспользуйтесь рабочей таблицей на рис. 19-7 для расчета затрат, необходимых для выполнения предполагаемых задач. Для выполнения большинства запросов на изменение потребуется только часть задач из рабочей таблицы, однако для выполнения других необходимы дополнительные задачи.

Затраты (Рабочие часы)	Задача
_____	Обновить спецификацию требований или базу данных требований.
_____	Разработать и оценить прототип.
_____	Создать новые компоненты дизайна.
_____	Изменить существующие компоненты дизайна.
_____	Разработать новые компоненты пользовательского интерфейса.
_____	Изменить существующие компоненты пользовательского интерфейса.
_____	Разработать новую пользовательскую документацию и экраны справки.
_____	Изменить существующую пользовательскую документацию и экраны справки.
_____	Разработать новый исходный код.
_____	Изменить существующий исходный код.
_____	Приобрести и встроить стороннее ПО.
_____	Изменить файлы сборки.
_____	Разработать новые тесты элементов и целостности продукта.
_____	Изменить существующие тесты элементов и целостности.
_____	Выполнить тестирование элементов и целостности продукта после реализации.
_____	Написать новые варианты тестирования системы и приемлемости.
_____	Изменить существующие варианты тестирования системы и приемлемости.
_____	Изменить автоматизированные механизмы тестирования.
_____	Выполнить регрессивное тестирование.
_____	Разработать новые отчеты.
_____	Изменить существующие отчеты.
_____	Разработать новые элементы базы данных.
_____	Изменить существующие элементы базы данных.
_____	Разработать новые файлы данных.
_____	Изменить существующие файлы данных.
_____	Изменить различные планы проекта.
_____	Обновить остальную документацию.
_____	Обновить матрицу трассируемости требований.
_____	Просмотреть измененные продукты.
_____	Переработать продукт после экспертизы и тестирования,
_____	Другие дополнительные задачи.
_____	**Всего затрат**

Рис. 19-7. Расчет затрат на изменение требования

4.Выполните расчет всех затрат.

5.Определите последовательность выполнения задач и то, как они могут пересекаться с уже запланированными задачами.

6.Определите, является ли изменение критичным для проекта. Если такая задача не будет выполнена в срок, дата завершения проекта сдвинется. На каждое изменение тратятся ресурсы, однако если вы можете запланировать изменение так, чтобы оно не влияло на критически важные в настоящее время задачи, то изменение не станет причиной нарушения графика.

7.Оцените влияние предложенного изменения на график и затраты.

8.Оцените приоритет изменения, выяснив относительные преимущества, неудобства, затраты и технический риск по сравнению с остальными требованиями.

9.Доложите о результатах анализа влияния совету по управлению изменениями, чтобы они смогли принять решение об утверждении или отклонении запроса на изменение.

Дополнительная информация

Что касается этапа 8, то в главе 14 описываются шкалы оценок для расстановки приоритетов требований, учитывающие преимущества, потери, затраты и риски.

В большинстве случаев эта процедура не должна отнимать более пары часов. Занятый разработчик, может, и считает, что это уж слишком, однако это небольшая плата за уверенность в том, что ограниченные ресурсы используются обдуманно. Если вы способны адекватно оценить влияние изменения без такой систематической процедуры — дерзайте, просто удостоверьтесь, что вы не идете по зыбучим пескам. Чтобы лучше оценивать влияние будущих изменений, сравните действительные затраты, необходимые для реализации каждого изменение, с рассчитанными затратами. Выясните причины всех различий и измените соответствующим образом списки оценки вопросов и рабочую таблицу.

Деньги на ветер

Два разработчика в компании A. Datum Corporation рассчитали, что уйдет четыре недели на добавление улучшения к одной из их информационных систем. Клиент утвердил их расчеты, и разработчики приступили к работе. Через два месяца работа была выполнена только наполовину, и клиент потерял терпение: «Если бы я знал, сколько времени это займет и сколько это будет стоить, я бы не согласился на это. Не нужно ничего делать». Торопясь получить одобрение и приступить к работе, разработчики не выполнили надежных расчетов анализа влияния, которые позволили бы клиенту принять должное бизнес-решение. В результате сотрудники A. Datum Corporation потратили сотни часов работу, которой можно было бы избежать, если бы они не поленились выделить всего несколько часов на предварительный анализ.

Шаблон отчета об анализе влияния изменения

На рис. 19-8 показан шаблон отчета о результатах анализа потенциального влияния каждого изменения требования. Стандартный шаблон облегчает членам совета по управлению изменениями поиск информации, необходимой для принятия правильных решений. Специалистам, реализующим изменение, понадобятся детали анализа и рабочая таблица планирования затрат, но совету требуется только итог результатов анализа. Как и любой другой шаблон, попробуйте его в работе, а затем измените в соответствии со спецификой вашего проекта.

ID запроса на изменение: _____
Название: _____
Описание: _____
Аналитик: _____
Дата подготовки запроса: _____
Расчеты приоритетов: Относительные преимущества: _____ (1-9) Относительные неудобства: _____ (1-9) Относительные затраты: _____ (1-9) Относительный риск: _____ (1-9) Рассчитанный приоритет: _____ (относительно др. рассматриваемых требований)
Расчет общих затрат: _____ рабочих часов
Расчет потерянных затрат: _____ рабочих часов (ненужная работа)
Расчет влияния на график: _____ дней
Дополнительное влияние на стоимость: _____ долларов
Влияние на качество: _____ _____
Другие требования, затрагиваемые изменением: _____
Другие задачи, затрагиваемые изменением: _____
Проблемы с целостностью: _____
Проблемы стоимости жизненного цикла: _____
Другие компоненты, которые необходимо проверить на предмет возможных изменений: _____ _____

Рис. 19-8. Шаблон отчета об анализе влияния

Изменение требований происходит на всех проектах, но хорошо организованные приемы контроля изменений уменьшают число проблем, возникающих из-за этого. Улучшенные приемы сбора информации по требованиям могут снизить количество изменений требований, а эффективное управление требованиями позволит вам более точно выполнять проектные обязательства.

Примечание

Рисунки с 19-5 по 19-8 можно найти по адресу *http://www.processimpact.com/goodies.shtml.*

Что теперь?

- Определите, кто будет принимать решения в вашем проекте, и назначьте их в совет по управлению изменениями. Примите устав, чтобы убедиться, что все члены совета понимают задачи совета, его состав и процесс принятия решений.
- Составьте диаграмму перехода состояний для жизненного цикла предложенных изменений требований на вашем проекте, начав с диаграммы на рис. 19-2. Опишите, как ваша команда будет обрабатывать предложенные изменения требований. Вначале делайте это вручную, пока не убедитесь, что процесс практичен и эффективен.
- Выберите коммерческое средство отслеживания проблем или вопросов, совместимое с вашей операционной средой разработки. Измените его в соответствии с процессом контроля изменений, созданном на предыдущем этапе.
- В следующий раз, когда вы будете оценивать запрос на изменение требований, сначала рассчитайте затраты, используя старый способ. Затем, оцените его еще раз, используя прием анализа влияния, описанный в этой главе. Если вы реализуете изменение, сравните результаты обоих расчетов, чтобы понять, какие расчеты точнее показывают непосредственные затраты. Измените списки вопросов для анализа влияния и рабочую таблицу согласно полученному опыту.

Глава 20 Связи в цепи требований

«Мы только что обнаружили, что в новом контракте с профсоюзом изменены правила расчета оплаты сверхурочных часов и надбавок за работу в ночную смену, — сообщил Джастин на еженедельном совещании. — Они также изменили то, как трудовой стаж влияет на график отпусков, на льготы и все остальное. Нам нужно обновить системы платежных ведомостей и графиков работы, чтобы обработать эти изменения, не откладывая. Крис, как ты думаешь, сколько времени это займет?»
«Ну, это большой объем работ, — сказал Крис. — Правила трудового стажа учитываются во всей системе составления графика, Я не могу назвать точный срок. У меня уйдут часы только на просмотр кода и попытку найти все фрагменты, где содержатся правила».

Простые изменения требования часто влияют на многие элементы продукта, поэтому возникает необходимость в их изменении. Трудно обнаружить все компоненты системы, которые могут быть затронуты модификацией требования. В главе 19 обсуждается важность анализа влияния — он позволяет убедиться, что команда понимает последствия изменения до принятия на себя обязательств по его выполнению. Выполнять анализ влияния проще, если вы построите ***дорожную карту*** (road map), на которой показано, где именно в ПО реализовано каждое требование и бизнес-правило.

В этой главе обсуждается контроль требований (или трассируемость). Трассируемость требований документирует зависимости и логические связи отдельных требований и других элементов системы. К этим элементам относятся требования различных типов, бизнес-правила, архитектура и другие компоненты дизайна, модули исходного кода, варианты тестирования и файлы справки. Информация трассируемости облегчает выполнение анализа влияния, помогая вам определить все рабочие продукты, которые вам может понадобиться изменить для реализации предложенного изменения требования.

Трассируемость требований

Связи трассируемости помогают следить за развитием требования в обоих направлениях — от первоисточника к реализации и наоборот (Gotel и Finkelstein, 1994). В главе 1 Трассируемость определяется, как одна из характеристик отличной спецификации требований. Чтобы реализовать Трассируемость, каждое требование должно быть уникально и последовательно идентифицировано, чтобы вы могли ссылаться на него однозначно в ходе работы над проектом. Пишите требования маленькими фрагментами, а не большими абзацами, в которых содержится множество отдельных функциональных требований — это приводит к разбросу элементов проектирования и кода.

Рис. 20-1. Четыре типа трассируемости требований

На рис. 20-1 показаны четыре типа связей трассируемости (Jarke, 1998). Потребности клиента отслеживаются *в* ***направлении к требованиям*** (forward to requirements), чтобы вы смогли определить, которые требования будут затронуты, если в течение или после разработки потребности изменятся. Это также дает уверенность, что в спецификации требований указаны все потребности клиента. И, наоборот, вы можете проследить в ***направлении от требований*** (backward from requirements) к потребностям клиента, чтобы определить происхождение каждого требования к ПО. Если вы представите потребности клиента в форме вариантов использования, то верхняя половина рис. 20-1 будет показывать трассирование между вариантами использования и функциональными требованиями.

Взглянув на нижнюю часть рис. 20-1, ясно, что по мере приближения к поставке, вы можете отслеживать *в направлении от требований,* определив связи между отдельными требованиями и элементами продукта. Это тип связи гарантирует, что каждое требование удовлетворено, поскольку вы знаете, какой компонент соответствует каждому требованию. Четвертый тип связи контролирует отдельные элементы продукта *в направлении к требованиям для* того, чтобы вы знали причину созданию каждого элемента. В большинство приложений включен код, не относящийся напрямую к требованиям пользователей, но вы должны знать, почему написана каждая строка кода.

Предположим, тестировщик обнаружит незапланированную функциональность при отсутствии соответствующего требования. Этот фрагмент кода может свидетельствовать, что разработчик реализовал официальное требование, которое аналитик теперь может добавить к спецификации. Или же это может быть код-сирота, украшающий фрагмент, который не относится к продукту. Связи трассируемости помогут вам отсортировать подобные ситуации и получить более полное представление о том, как именно фрагменты вашей системы составляют одно целое. И наоборот, варианты тестирования, которые созданы на основе отдельных требований и которые можно проследить до этих требований, также представляют собой механизм выявления нереализованных требований, поскольку ожидаемой функциональности не будет.

Связи трассируемости помогают отслеживать родительские требования, взаимосвязи и зависимости между отдельными требованиями, Эта информация показывает влияние изменения, если отдельное требование удаляется или модифицируется. Если вы сопоставили отдельные требования с задачами в структуре проекта, то эти задачи будут затронуты при изменении или удалении требования.

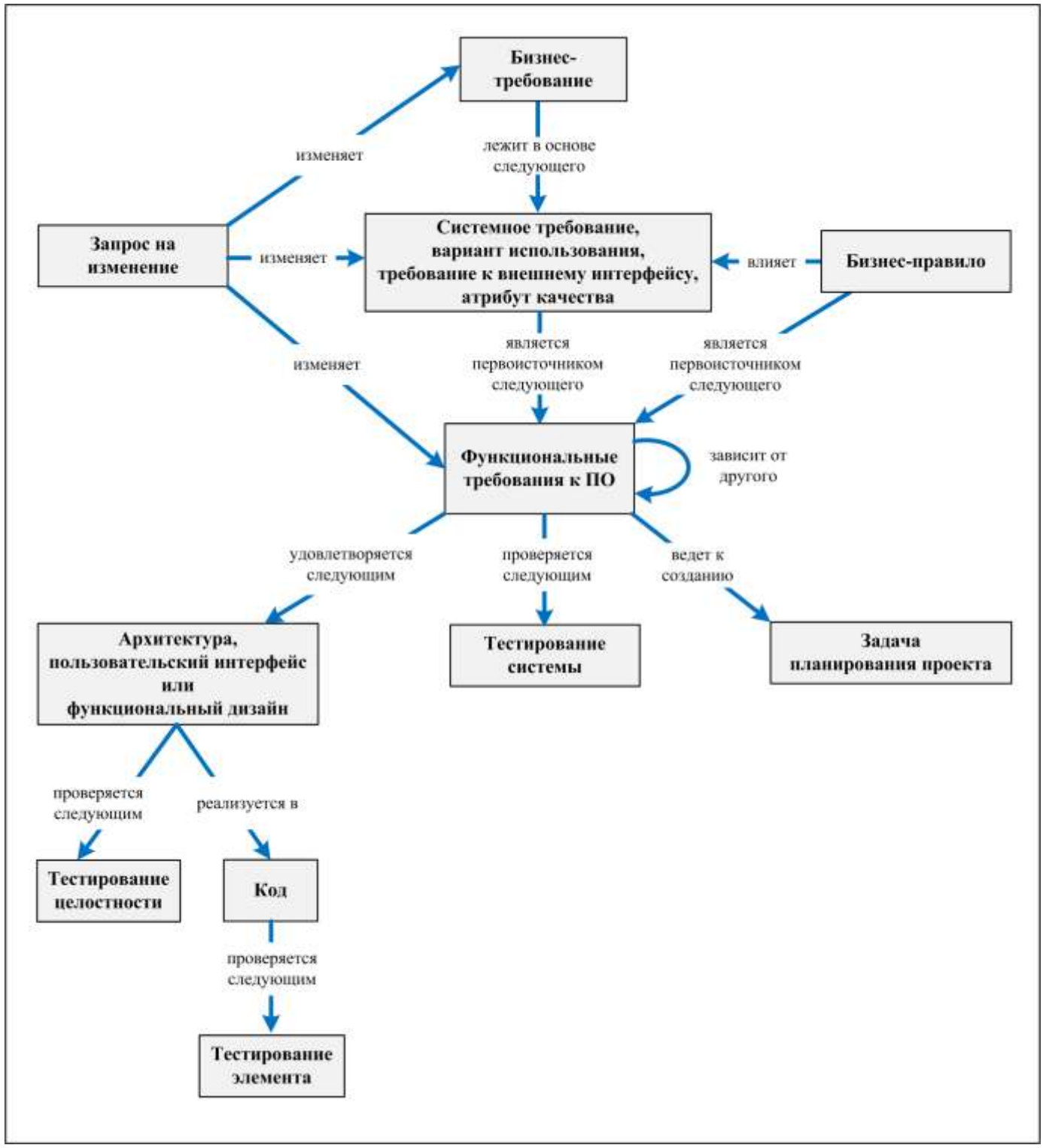


Рис. 20-2. Возможные связи трассирования требований

На рис. 20-2 показано множество типов прямых взаимосвязей трассируемости, возможных в проекте. Вам не нужно определять и управлять всеми этими типами связей трассируемости. Во многих проектах удается получить 80% преимуществ трассируемости, вложив приблизительно 20% усилий. Возможно, вам нужно только отслеживать тестирование системы до функциональных требований или вариантов использования. Решите, какие связи уместны в вашем проекте, и вы в значительной степени поспособствуете успешной разработке и эффективному обслуживанию. Не просите членов команды тратить много времени на фиксирование информации, если у вас нет четкого представления о использовании этой информации.

Мотивация для трассируемости требований

Однажды у меня был неудачный опыт: я написал программу, а потом понял, что случайно упустил из виду требование. Оно было в спецификации — я просто не заметил его. Мне пришлось вернуться и написать дополнительный код, а я думал, что уже закончил с программированием. Пропуск требования — это нечто более, чем просто промах, если клиент не удовлетворен или в готовом продукте отсутствует функция, особо важная для обеспечения безопасности. На одном уровне трассирование требований предоставляет способ продемонстрировать соответствие контракту, спецификации или правилу. На более сложном уровне это позволяет улучшить качество продуктов, снизить затраты на поддержку и облегчить повторное использование (Ramesh, 1998).

Трассирование требований — это трудоемкая задача, выполняемая вручную, для которой необходимо организационное согласование. Для обновления информации о связях по мере разработки и обслуживания системы необходима дисциплина. Если информация трассируемости устаревает, вероятнее всего, вам никогда не удастся заново воссоздать ее. Кроме того, из-за этого разработчики и сотрудники службы поддержки напрасно тратят время, следуя по неверному пути. Именно поэтому для использования трассируемости у вас должна быть веская причина (Ramesh и др., 1995). Далее перечислены возможные преимущества реализации трассируемости требований.

- **Сертификация.** Вы можете воспользоваться информацией трассируемости при сертификации продукта с особыми требованиями к безопасности, чтобы продемонстрировать, что все требования были реализованы — хотя это не доказывает, что они реализованы корректно и полностью! Естественно, если требования некорректны или отсутствуют ключевые требования, то вам не помогут даже самые лучшие данные трассируемости.
- **Анализ влияния изменения.** Без информации трассируемости возрастает вероятность того, что из внимания будет упущен системный элемент, которого коснется добавление, удаление или изменение определенного требования.
- **Поддержка.** Надежная информация трассируемости облегчает соответствующее и полное внесение изменений в ходе обслуживания, что повышает производительность. Часто при изменении корпоративных политик или государственных правил, приложения ПО нуждаются в обновлении. Таблица, в которой показано, где соответствующее бизнес-правило было реализовано в функциональном требовании, проектировании и коде, упрощает соответствующее внесение изменений.
- **Трассирование проекта.** Если в ходе разработки вы тщательно фиксируете данные трассируемости, у вас будет точное представление о состоянии реализации запланированной функциональности. Отсутствующие связи указывают на рабочие продукты, которые еще не созданы.
- **Повторная разработка.** Вы можете перечислить функции в предыдущей версии системы, которую вы заменяете, и указать, каким новым системным требованиям и компонентам ПО они адресованы. Определение связей трассируемости это способ зафиксировать определенный опыт, полученный при переработке существующей системы.
- **Повторное использование.** Информация трассируемости упрощает повторное использование компонентов продукта, определяя пакеты связанных требований, проекта, кода и тестов.
- **Снижение риска.** Документирование взаимосвязей компонентов уменьшает риск возникновения проблем, если вдруг ключевой член команды, обладающей важной информацией о системе, покидает проект (Ambler, 1999).
- **Тестирование.** Если тестирование дает неожиданный результат, то связи между тестами, требованиями и кодом укажут на наиболее вероятные части кода, которые необходимо проверить на наличие дефектов. Информация о том, какие тесты проверяют какие требования, экономит время, позволяя вам удалить лишние тесты. Многие из вышеперечисленных относятся к долгосрочным выгодам, которые снижают

общую стоимость жизненного цикла продукта, хотя при этом увеличиваются затраты на сбор и управление информации трассируемости. Рассматривайте трассируемость требований как инвестицию, увеличивающую шансы поставить продукт, который удовлетворяет все основные требования клиента. Хотя это трудно измерить, вы будете получать дивиденды каждый раз, когда вам понадобится изменить, расширить или заменить продукт. Определить связи трассируемости несложно, если вы собираете информацию по мере разработки, однако делать это для уже завершенной системы утомительно и дорого.

Матрица трассируемости требований

Наиболее типичный способ представления связей между требованиями и другими элементами системами — *матрица трассируемости требований,* которую также называют ***матрицей отслеживания требований*** или ***таблицей трассируемости*** (requirements traceability matrix) (Sommerville и Sawyer, 1997). В табл. 20-показана часть такой матрицы для Chemical Tracking System. Когда я раньше создавал такие матрицы, я делал копию базовой версии спецификации требований и удалял все, кроме идентификаторов функциональных требований. Затем я создавал таблицу, отформатированную так же, как табл. 20-1, и заполнял только столбец Функциональное требование. Далее мы постепенно заполняли пустые ячейки в матрице по мере разработки проекта.

Таблица 20-1. Один из типов матрицы трассируемости требований

Пользовательское требование	Функциональное требование	Элемент проектирования	Модуль кода	Вариант тестирования
UC-28	catalog.query.sort	Каталог класса.	catalog.sort()	search. 7 search. 8
UC-29	catalog.query.import	Каталог класса	catalog.import() catalog.validate()	search. 12 search. 13 search. 14

В табл. 20-1 показано, как каждое функциональное требование связано с определенным вариантом использования (в направлении назад) и с одним или более элементами проектирования, кода и тестирования (в направлении вперед). Элементы дизайна могут быть объектами в таких моделях анализа, как диаграммы потока данных, таблицы в реляционной модели данных или классах объектов. Ссылки кода могут быть методами класса, хранимыми процедурами, именами файлов исходного кода, а также процедурами или функциями в исходном файле. Вы вправе добавить дополнительные столбцы для расширения ссылок на другие рабочие продукты, например на документацию справочной системы. Добавление деталей трассируемости - дополнительная работа, но так вы получаете точные расположения связанных элементов ПО, что экономит массу времени при анализе влияния и обслуживании.

Заполняйте информацию по мере выполнения работы, а не по мере планирования. То есть вводите «catalog.sort()» в столбец Модуль кода первой строки в табл. 20-1 только когда код в этой функции написан, прошел тестирование элементов и уже интегрирован с базовой версией исходного кода продукта. Таким образом, читающий матрицу будет знать, что заполненный ячейки матрицы для отслеживания требований указывают на работу, которая уже выполнена. Обратите внимание, что перечисление вариантов тестирования для каждого требования ***не указывает*** на то, что ПО протестировано. Это просто означает, что определенные тесты были написаны для проверки требований в соответствующее время. Трассирование состояния тестов— это отдельная проблема.

Нефункциональные требования, такие, как задачи по производительности и атрибуты качества не всегда прослеживаются напрямую до кода. Требование к времени отклика может диктовать выбор определенного оборудования, алгоритмов, структур баз данных или архитектуры.

Требование к легкости перемещения может ограничить функции языка, используемые программистом, однако не приведет к созданию определенных фрагментов кода, которые активизируют эту возможность. Другие же атрибуты качества действительно реализуются в коде. Требования к целостности для аутентификации пользователей активизирует создание производных функциональных требований, которые реализуются, с помощью, скажем, паролей или биометрических параметров. В этих случаях следует трассировать соответствующие функциональные требования в обратном направлении, к их родительским нефункциональным требованиям, и, как обычно, в прямом, до готового продукта. На рис. 20-3 показана возможная цепь трассируемости с участием нефункциональных требований.

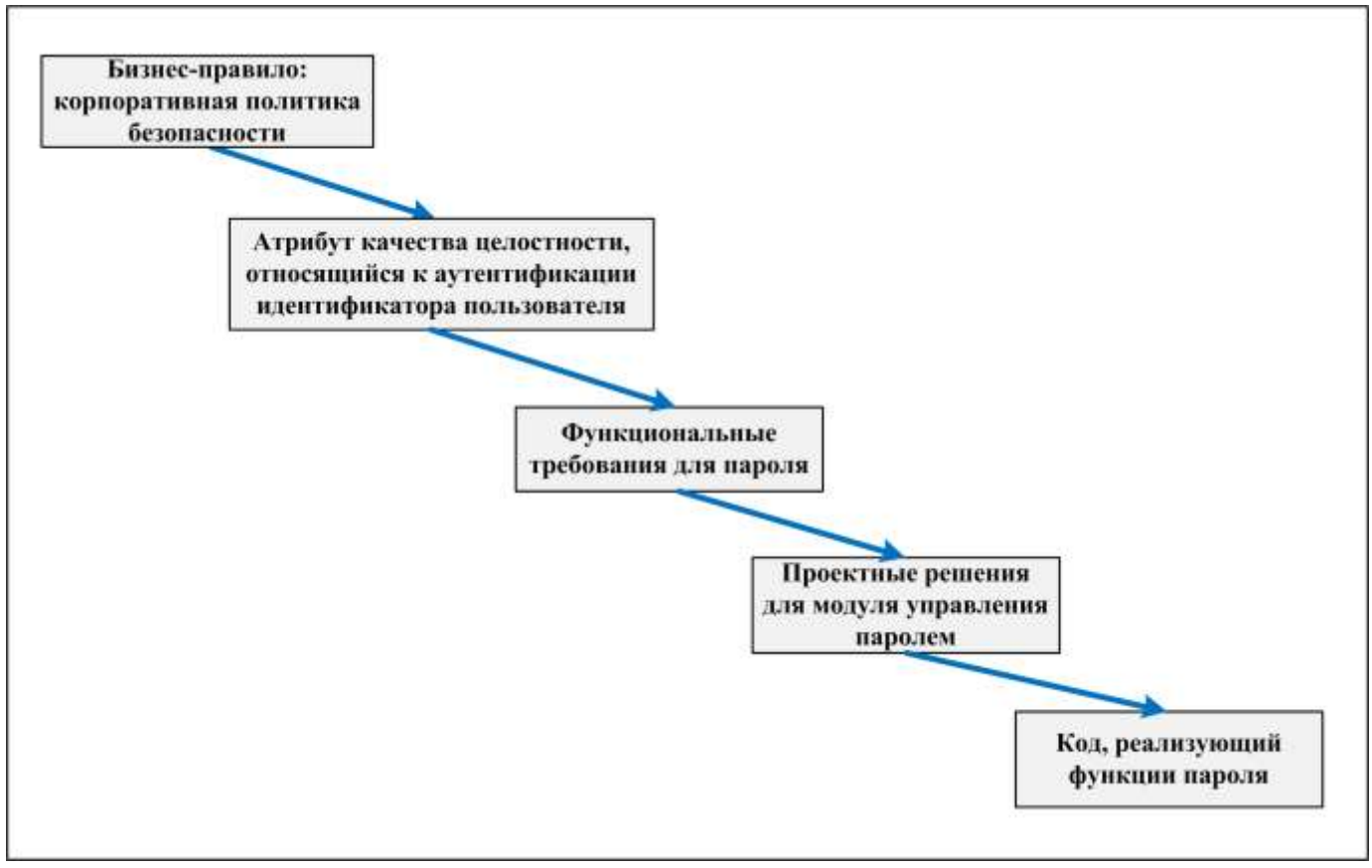


Рис. 20-3. Пример цепи трассируемости для требований, касающихся безопасности приложения

Связи трассируемости могут определить отношения «один к одному», «один ко многим» или «многие ко многим» между элементами системы. Формат в табл. 20-1 предусматривает это, позволяя вводить несколько позиций в каждой ячейке таблицы. Ниже приведены примеры возможных связей.

- **«Один к одному»:** один элемент проектирования реализуется в одном модуле кода.
- **«Один ко многим»:** одно функциональное требование проверяется множеством вариантов тестирования.
- **«Многие ко многим»:** каждый вариант использования порождает множество функциональных требований, а определенные функциональные требования являются общими для нескольких вариантов использования. Подобным образом общие или повторяющиеся элементы проектирования могут удовлетворить несколько функциональных требований. В идеале стоило бы зафиксировать все эти взаимосвязи, однако на практике отношениями трассируемости типа «многие ко многим» сложно и трудно управлять.

Другой способ представить информацию трассируемости — с помощью набора матриц, определяющих связи между парами элементов системы, например:

- один тип требования с другим требованием этого же типа;

- один тип требования с требованием другого типа;
- один тип требования с вариантами тестирования.

Вы можете использовать эти матрицы для определения различных взаимоотношений, возможными между парами требований, например «указывает/указан», «зависит от», «является родительским для» и «ограничивает/ограничен» (Sommerville и Sawyer, 1997).

В табл. 20-2 показана двусторонняя матрица трассируемости. Большинство ячеек матрицы не заполнены. Каждая ячейка на пересечении двух связанных компонентов помечена для указания соединения. Вы можете использовать различные символы в ячейках, чтобы явно определить «трассируется до» и «трассируется от» или другие взаимоотношения. В табл. 20-2 стрелка указывает, что данное функциональное; требование отслеживается от определенного варианта использования. Эти матрицы более поддаются автоматизации средствами поддержки, чем те, что показаны в табл. 20-1.

Таблица 20-2. Матрица для трассирования требований, показывающая связи между вариантами использования и функциональными требованиями

	Вариант использования (ВИ)			
Функциональное требование (ФТ)	**ВИ-1**	**ВИ-2**	**ВИ-3**	**ВИ-4**
ФТ-1	↵			
ФТ-2	↵			
ФТ-3			↵	
ФТ-4			↵	
ФТ-5.		↵		↵
ФТ-6			↵	

Связи трассируемости должны быть определены любым лицом, имеющим доступ к соответствующей информации. В табл. 20-3 определены некоторые стандартные источники информации о связях между различными типами исходных и целевых объектов. Определите роли и лиц, которые будут поддерживать каждый тип информации трассируемости для вашего проекта. Будьте готовы к тому, что занятые люди, которых аналитик или менеджер проекта попросит предоставить эти данные, будут отнекиваться. Этим специалистам стоит объяснить, что такое трассирование требований, чем оно ценно и почему именно этих специалистов просят внести вклад в процесс. Подчеркните, что увеличение затрат на фиксирование информации трассируемости во время выполнения невелики; в основном это вопрос привычки и дисциплины.

Ловушка

За сбор и управление данными трассируемости требований должны Отвечать определенные лица, или оно просто не будет выполнено. Обычно аналитик требований или специалист по проверке соответствия качества собирает, сохраняет информацию такого рода и составляет отчеты по информации трассируемости.

Таблица 20-3. Вероятные источники информации о связи трассируемости

Тип объекта источника ссылки	Тип объекта целевой ссылки	Источник информации
Системное требование	Требование к ПО	Системный инженер
Вариант использование	Функциональное требование	Аналитик требований
Функциональное требование	Функциональное требование	Аналитик требований
Функциональное требование	Вариант тестирования	Специалист по тестированию
Функциональное требование	Элемент архитектуры ПО	Архитектор ПО
Функциональное требование	Другие элементы проектирования	Проектировщик или разработчик
Элемент проектирования	Код	Разработчик
Бизнес-правило	Функциональное требование	Аналитик требований

Средства трассирования требований

В главе 21 описано несколько коммерческих средств управления требованиями, обладающих значительными возможностями трассирования требований. Вы можете сохранить требования и другую информацию в базе данных средства и определить связи между различными типами сохраненных объектов, включая равноценные ссылки между двумя требованиями одного типа. Некоторые инструменты позволят вам различить отношения «трассируется до» и «трассируется от», автоматически определяя дополнительную ссылку. То есть, если вы укажете, что требование R отслежено до варианта тестирования T, средство также покажет симметричное отношение, в котором T трассируется от R.

Некоторые средства автоматически каждый раз помечают ссылку ***как подозрительную,*** когда объект на одном из концов связи изменяется. Подозрительная ссылка помечается (например, знаком вопроса красного цвета или диагональной линией красного цвета) в соответствующей ячейке матрицы трассирования требований. Например, если вы измените вариант использования 3, то матрица трассирования требований в табл. 20-2 может выглядеть, как показанная в табл. 20-4, когда вы на нее посмотрите в следующий раз. Указатель подозрительной связи (в данном случае знак вопроса) напоминает вам, что нужно проверить, следует ли модифицировать функциональные требования 3, 4 и 6, чтобы те остались совместимыми с измененным ВИ-3. После внесения всех необходимых изменений необходимо вручную убрать указатели подозрительных ссылок. Это процесс помогает убедиться, что вы учли все возникшие в результате волнового эффекта изменения.

Таблица 20-4. Подозрительные связи в матрице трассирования требований

	Вариант использования (ВИ)			
Функциональное требование (ФТ)	**ВИ-1**	**ВИ-2**	**ВИ-3**	**ВИ-4**
ФТ-1	↵			
ФТ -2	↵			
ФТ -3			↵?	
ФТ -4			↵?	
ФТ -5.		↵		↵
ФТ -6			↵?	

Средства также позволяют определить связи между проектами или между подсистемами. Я знаю об одном крупном проекте ПО с 20 крупными подсистемами, где требования к продукту высокого уровня распределены среди этих подсистем. В некоторых случаях требование, адресованное одной подсистеме, реализовалось с помощью службы, предоставленной другой подсистемой. В этом проекте для успешного трассирования этих сложных отношений трассирования использовалось средство управления требованиями.

Трассирование требований вручную невозможно выполнить ни для одного приложения, за исключением самых маленьких. Вы можете воспользоваться рабочей таблицей для поддержания данных трассируемости почти пары сотен требований, однако для более крупных систем необходимо решение понадежнее. Трассирование требований нельзя полностью автоматизировать, поскольку данные об источниках связей хранятся в головах разработчиков. Однако после того как вы определите связи, средства помогут вам управлять огромным объемом информации трассируемости.

Процедура трассирования требований

Придерживайтесь следующей последовательности действий для реализации трассирования требований,

1.Выберите отношение связей, которое вы хотите определить, из тех, что показаны на рис. 20-2.

2.Выберите тип матрицы трассирования требований, которую вы хотите использовать: показанную в табл. 20-1 или в табл. 20-2. Выберите механизм для хранения данных: таблицу в текстовом документе, рабочую таблицу или средство управления требованиями.

3.Определите части продукта, для которой вы хотите поддерживать информацию трассируемости. Начните с важнейших основополагающих функций, фрагментов, отличающихся высоким риском, или частей, которые, как вы и ожидали, потребуют большей части поддержки и будут быстрее развиваться в течение жизненного цикла продукта.

4.Модифицируйте процедуры разработки и списки вопросов, чтобы напомнить разработчикам об обновлении связей после реализации требования или утверждения изменения. Данные трассируемости должны обновляться сразу после завершения задачи, в результате которой создается или изменяется звено в цепи требований,

5.Определите соглашения об именовании, которые вы будете использовать для уникальной идентификации всех элементов системы таким образом, чтобы их удалось связать друг с другом (Song и др., 1998). При необходимости напишите сценарии, которые будут выполнять анализ файлов системы для сборки и обновлять матрицы трассирования.

6.Определите лиц, которые будут поддерживать каждый тип информации о связях, и сотрудника, координирующего трассирование и управляющего данными.

7.Проинформируйте команду о концепции и важности трассирования требований, о целях этого, о том, где хранятся данные трассируемости, и приемах для определения связей (например, с помощью функции трассирования средства управления требованиями). Убедитесь, что все участники осознают их ответственность.

8.Попросите каждого участника предоставлять запрашиваемую информацию трассируемости по мере завершения небольших участков работы. Напоминайте о том, что работа над созданием данных трассируемости не прекращается ни на минуту; не говорите о том, что вы попытаетесь воссоздать эти данные на важном этапе работы или в конце проекта.

9.Периодически проверяйте информацию трассируемости, чтобы убедиться, что она не устарела. Если поступает сообщение, что требование реализовано и проверено, но данные трассируемости неполны или неточны, означает, что процесс трассирования требований не работает так, как ожидалось.

Я описал эту процедуру, как если бы вы начинали собирать информацию трассируемости с начала нового проекта. Если вы обслуживаете действующую систему, то бьюсь об заклад, что у вас нет доступных данных трассируемости, однако у вас нет времени начинать сбор этой полезной

информации. В следующий раз, когда вы добавите улучшение или внесете изменение, запишите все, что вы узнали в связях между кодом, тестами, проектом и требованиями. Учтите необходимость трассируемости данных в процедуре для изменения существующего компонента ПО. Вам уже не удастся собрать полную матрицу трассирования требований, однако это небольшое усилие может облегчить работу над этой же частью системы в следующий раз.

Осуществимость и необходимость трассирования требований

Вы можете прийти к заключению, что на создание матрицы трассирования требований вы тратите больше, чем получаете, или что она неосуществима для вашего крупного проекта, поэтому я хочу привести вам контр пример. Как-то на конференции один участник, занимающийся строительством самолетов, сказал мне, что спецификация требования для работы над последним реактивным самолетом представляла собой кипу бумаги высотой в шесть футов. У них была полная матрица трассирования требований. Я летал на именно этой модели самолета и был счастлив узнать, что разработчики так тщательно поработали над своими требованиями к ПО. Управление трассируемостью для большого продукта с множеством взаимосвязанных подсистем — это большой объем работы. Мой новый знакомый знал, что это очень важное дело, и Федеральное управление гражданской авиации было с ним согласно.

Даже если в случае неудачи ваши продукты не могут стать причиной угрозы жизни или здоровью людей, вам следует серьезно отнестись к трассированию требований. Когда я рассказывал о трассируемости на семинаре, глава крупной корпорации спросил: «Почему вы не создадите матрицу трассирования требований для вашей стратегической бизнес-системы?» Это отличный вопрос. Вы должны принимать решение об использовании любых улучшенных приемов разработки требований, принимая во внимание и затраты на внедрение приема и риск отказа от приема. Как и со всеми процессами ПО, экономические факторы должны определять, тратить ли ваше ценное время там, где возможна наибольшая отдача.

Что теперь?

- Создайте матрицу трассирования требований для 15 или 20 требований к важной части системы, которую вы в настоящее время разрабатываете. Опробуйте приемы, показанные в табл. 20-1 и в табл. 20-2. Заполняйте матрицу по мере разработки, в течение нескольких недель. Оцените, какой метод кажется наиболее эффективным и какие процедуры для сбора и хранения информации трассируемости подойдут для вашей команды,
- В следующий раз, когда будете обслуживать плохо задокументированную систему, записывайте, чему научил вас инженерный анализ продукта, который вы модифицируете. Постройте фрагмент матрицы трассирования требований для фрагмента головоломки, над которой вы сейчас работаете, чтобы облегчить жизнь тому, кто займется ей в следующий раз. Расширяйте матрицу трассирования требований по мере обслуживания проекта.

Глава 21 Инструментальные средства управления требованиями

В предыдущих главах я уже говорил о создании спецификации требований к ПО на естественном языке, которая содержит функциональные и нефункциональные требования, а также, о создании документов, содержащих бизнес-требования и описания вариантов использования. Документальный способ хранения требований имеет массу ограничений, в том числе:

- трудность обновления и синхронизации документов;
- всем членам команды, которым это необходимо, передачу изменений приходится осуществлять вручную;
- трудность хранения дополнительной информации (атрибутов) для каждого требования; трудность определения взаимосвязей между функциональными требованиями и другими элементами системы;
- трассирование статуса требований представляет собой трудный и неудобный процесс;
- одновременное управление наборами требований, запланированных для различных выпусков или взаимосвязанных продуктов, затруднено. Когда реализация требования откладывается до следующего выпуска, аналитику приходится перемещать его из одной спецификации требований в другую;
- при повторном использовании какого-то требования аналитик приходится копировать текст исходной спецификации требований в спецификацию требований каждой системы или продукта, где оно будет использоваться;
- трудность модификации требований несколькими участниками проекта, особенно если они географически разделены;
- отсутствие удобного места хранения требований, которые были предложены, но отклонены, или требований, удаленных из основной версии.

Средство управления требованиями, хранящее информацию в многопользовательской базе данных, позволяет снять эти ограничения. В небольших проектах для управления требованиями вам пригодятся электронные таблицы или простые базы данных, сохраняющие как текст, так и отдельные атрибуты каждого требования. В более крупных проектах выгодно применять коммерческие средства управления требованиями. Такие продукты позволяют пользователям импортировать требования из исходных документов, определять значения атрибутов, фильтровать и выводить на экран содержание базы данных, экспортировать требования в различных форматах, контролировать связи трассирования и соединять требования с элементами, хранящимися в других средствах разработки ПО.

Ловушка

Избегайте искушения разработать собственное средство управления требованиями или на скорую руку слепить его из средств офисной автоматизации общего назначения в подражание коммерческим продуктам. Первоначально это кажется легко, но такая работа может быстро истощить ресурсы команды, не обладающей средствами, необходимыми для создания желаемого инструмента.

Обратите внимание, я называю эти продукты средствами ***управления*** требованиям, а не их ***разработки***. Они не помогут вам определить потенциальных пользователей или собрать нужные требования для создаваемого продукта. Тем не менее они позволяют очень гибко управлять изменениями в этих требованиях и использовать их как основу для конструирования, тестирования и управления продуктом. Эти инструментальные средства не заменяют собой приемов, с помощью которых члены вашей команды выявляют требования и управляют ими. Используйте инструменты, когда у вас уже есть рабочая методика, но ей не хватает эффективности; не

надейтесь, что инструмент восполнит недостаток трудолюбия, дисциплины, опыта или понимания.

Данная глава рассказывает о нескольких преимуществах использования средств управления требованиями и указывает на некоторые общие возможности таких продуктов. В табл. 21-1 перечислено несколько доступных в настоящее время инструментальных средств управления требованиями. В этой главе я не буду подробно сравнивать их — функция за функцией, поскольку эти продукты еще развиваются, и их возможности изменяются с каждой версией. Цены на них, платформы, которые они поддерживают, и даже производители также часто изменяются, поэтому для получения текущей информации используйте Интернет-адреса, приведенные в табл. 21-1 (с учетом того, что и Интернет-адреса могут изменяться, если, например, один производитель ПО поглотит другого, как случилось за две недели до написания этого материала). Детализированное сравнение возможностей этих и многих других инструментальных средств вы найдете на Web-странице Intelnational Council on Systems Engineering *(http://www.incose.org/toc.html- **Error 404: Not Found прим. Редактора)**.* Там же опубликованы рекомендации по выбору инструментального средства управления требованиями (Jones и др., 1995).

Таблица 21-1. Некоторые коммерческие инструментальные средства управления требованиями

Инструмент	Производитель	Способ хранения данных
Active! Focus	Xapware Technologies, *http://www.hapware.com*	База данных
CaliberRM	Borland Software Corporation, *http://www.borland.com*	База данных
C.A.R.E,,	SOPHIST Group, ***http://www.sophist.de***	База данных
IDOORS	Telelogic, ***http:/www.telel ogic.com***	База данных
RequisitePro	Rational Software Corporation, ***http://www.rational.com***	Документ
RMTrak	RBC, Inc., ***http://www2.rbcorp.com***	Документ
RTM Workshop	Integrated Chipware, Inc. ***http://www.chipware.com***	База данных
Slate	EDS, ***http://www.eds.com***	База данных
Vital Link	Compliance Automation, Inc., ***http://www.complianceautomation.com***	Документ

Важное отличие инструментальных средств заключается в способе хранения данных. Одни хранят все требования, атрибуты и информацию трассирования в базе данных. В зависимости от продукта, база данных может быть коммерческой или разработанной собственными силами и запатентованной, реляционной или объектно-ориентированной. Требования могут импортироваться из различных исходных документов, но затем они сохраняются в базе данных. Как правило, текстовое описание требования считается просто одним из атрибутов. Некоторые продукты позволяют устанавливать связи отдельных требований с внешними файлами (например, файлами Microsoft Word, Microsoft Excel, графическими файлами и т.д.), в которых содержится информация, дополняющая содержимое базы.

При документальном подходе документ, созданный при помощи текстового процессора (такого, как Microsoft Word или Adobe FrameMaker), считается основным хранилищем требований. RequisitePro позволяет вам выбирать текстовые строки в документе Word для сохранения их в виде отдельных требований в базе данных. После ввода требований в базу данных вы можете определить атрибуты и связи трассирования так же, как и в продуктах, данные которых хранятся в БД — механизмы синхронизации базы данных и содержимого документа для этого предусмотрены. RTM Workshop поддерживает обе схемы: в первую очередь хранение данных в базе данных, но также и в документе Microsoft Word.

Эти средства не дешевы, но высокая стоимость решения проблем управления требованиями

оправдывает инвестиции в них. Учитывайте, что цена инструментального средства определяется не только ценой лицензии. Вам придется потратиться на компьютер, на который будет установлена программа, учесть ежегодные затраты на поддержку и периодическое обновление продукта, на установку ПО, администрирование, техническую поддержку и консультации производителя, а также на обучение пользователей. Проанализируйте возможные затраты и результаты до того, как примете решение о покупке.

Преимущества использования инструментальных средств управления требованиями

Даже если вы провели внушительную работу по сбору требований для своего проекта, автоматизированная поддержка поможет вам оперировать с этими требованиями в процессе разработки. Инструментальное средство управления требованиями становится все полезнее со временем, когда детали требований постепенно тускнеют в памяти членов команды. В следующих разделах описаны некоторые задачи, которые подобное средство поможет вам решать.

Управление версиями и изменениями. Ваш проект должен определять основную версию требований — четкий набор требований для конкретной версии. Некоторые средства управления требованиями предлагают функции гибкого управления базовой версией. Они также сохраняют историю изменений каждого требования. Вы можете записывать обоснование каждого решения об изменении и при необходимости возвратиться к предыдущей версии требования. Некоторые средства, такие как Active! Focus и DOORS, содержат простые, встроенные системы изменений-предложений, устанавливающие связи между предложениями об изменениях и измененными требованиями.

Хранение атрибутов требований. Вы должны записывать несколько описательных атрибутов для каждого требования, как говорилось в главе 18. Каждый, кто работает над проектом, должен иметь доступ к просмотру этих атрибутов, а некоторые — к изменению их значений. Инструментальное средство управления требованиями генерирует несколько системных атрибутов, например дату создания требования и номер его версии, а также, позволяет вам создавать дополнительные атрибуты различных типов данных. Продуманное определение атрибутов позволяет всем заинтересованным в проекте лицам просматривать подмножества требований, основанных на выбранных комбинациях значений атрибутов. Вы можете запросить список всех требований, основанных на каком-либо бизнес-правиле, чтобы принять решение о последствиях изменения этого правила. Один из способов учета требований в основных версиях различных выпусков продукта — использовать атрибут «номер выпуска».

Облегчение анализа воздействия. Средства управления требованиями помогают осуществлять трассирование требований, позволяя вам определять связи между различными типами требований, между требованиями в различных подсистемах и между отдельными требованиями и связанными системными компонентами (например, проектным решением, модулями кода, тестами и пользовательской документацией). Эти связи помогут вам анализировать воздействие, которое предлагаемое изменение окажет на конкретное требование, выявляя другие элементы системы, которые оно затронет. Другой полезный способ — отследить каждое функциональное требование назад до первоисточника, чтобы знать, откуда оно берет начало.

> **Дополнительная информация**
>
> В главе 19 описан анализ воздействия, а в главе: 20 — трассирование требований.

Трассирование статусов требований. Собрав требования в базе данных, вы узнаете, сколько отдельных требований вы указали для продукта. Трассирование статуса каждого требования в процессе разработки способствует общему трассированию статуса проекта. Менеджер проекта точно представляет себе статус проекта, если знает, что 55% требований для следующего выпуска проверены, 28% — реализованы, но не проверены, а 17% еще не до конца реализованы.

Контролируемый доступ. Средства управления требованиями позволяют вам устанавливать

права доступа для отдельных людей и групп пользователей и предоставлять информацию в общий доступ географически распределенной команде через Web-интерфейс к базе данных. Базы данных используют блокировку на уровне требований, позволяя сразу многим пользователям обновлять содержимое базы данных одновременно.

Связь со всеми заинтересованными в проекте лицами. Некоторые средства управления требованиями позволяют членам команды обсуждать вопросы, связанные с требованиями, на тематических дискуссиях с помощью электронных средств обмена сообщениями. Автоматически отсылаемые электронные сообщения уведомляют членов команды, когда в дискуссии появляется новая запись или любое требование модифицируется. Возможность работы с требованиями через Интернет сокращает расходы на транспорт и уменьшает документооборот.

Повторное использование требований. Хранение требований в базе данных облегчает их повторное использование в других проектах и подпроектах. Требования, логически подходящие нескольким разделам описания проекта, можно сохранить однажды, а затем лишь ссылаться на них во избежание дублирования требований.

Возможности инструментальных средств управления требованиями

Коммерческие средства управления требованиями позволяют вам определять различные типы (или классы) требований, такие, как бизнес-требования, варианты использования, функциональные требования, требования к оборудованию, а также ограничения. Это позволяет вам отделять задачи, с которыми можно работать, как с требованиями, от другой полезной информации, содержащейся в спецификации требований к ПО. Все инструментальные средства обладают мощными возможностями определения атрибутов для каждого типа требований, что представляет огромное преимущество перед обычными способами документирования спецификации требований к ПО.

Большинство инструментальных средств управления требованиями в той или иной степени интегрируются с Microsoft Word — на панель меню Microsoft Word добавляется их специализированный раздел. Vital Link основан на Adobe FrameMaker, а Slate интегрируется и с FrameMaker, и с Word. Инструментальные средства высокого уровня поддерживают широкий набор форматов файлов для импорта и экспорта. Некоторые инструменты позволяют пометить текст в документе Word, чтобы обращаться с ним, как с отдельным требованием. Инструмент выделяет требование и вставляет в документ закладки Word и скрытый текст. Кроме того, вы получаете возможность различными способами анализировать документы, чтобы извлекать из них отдельные требования. Анализ документа в текстовом редакторе будет несовершенным, если только при создании документа вы не станете последовательно использовать стили или ключевые слова, такие как «должно».

Инструментальные средства поддерживают иерархическую систему нумерации требований помимо уникального внутреннего идентификатора для каждого требования. Эти идентификаторы обычно состоят из короткого текстового префикса, который указывает на тип требования, например UR обозначает ***требование пользователя*** (user requirement), и уникального целого числа. Некоторые средства предоставляют возможность пользоваться Microsoft Windows Explorer — для отображения и управления иерархическим деревом требований. Один из способов отображения требований в DOORS — иерархически структурированная спецификация требований к ПО.

К возможностям вывода данных инструментальных средств относится способность генерировать требования либо в документе заданного пользователем формата, либо в табличном отчете. CaliberRM обладает мощной функцией Document Factory, позволяющей определить шаблон спецификации требований к ПО в Word, используя простые способы для разметки макета страницы, шаблона текстов, атрибутов для извлечения данных из базы и стилей текста, которые вы хотите применять. Document Factory заполняет этот шаблон информацией, которую выбирает из базы данных соответственно критериям заданного пользователем запроса, чтобы создать заказанный документ спецификации требований к ПО. Таким образом, спецификация требований

к ПО, в сущности, представляет собой отчет, генерируемый по некоторой выборке из содержимого базы данных.

Все инструментальные средства обладают развитыми возможностями трассирования. Например, в RTM Workshop каждый проект определяет схему классов, похожую на диаграмму «сущность-связь», для всех хранящихся типов объектов. Трассирование выполняют, определяя связи между объектами двух классов (или внутри одного класса) на основе взаимоотношений между классами, заданными в схеме.

К числу других функций относится возможность задавать группы пользователей и определять разрешения некоторым пользователям или группам на создание, чтение, обновление и удаление проектов, требований, атрибутов и их значений. Некоторые продукты позволяют поместить нетекстовые объекты, такие, как графику и крупноформатные таблицы, в базу требований. Инструментальные средства предлагают также средства обучения или проекты примеры, которые помогут пользователям научиться работать быстро и эффективно.

Эти продукты отражают тенденцию к увеличению интеграции с другими инструментальными средствами, используемыми в разработке приложений, как показано на рис. 21-1. Выбирая средство для управления требованиями, выясните, сможет ли оно обмениваться данными с другими используемыми вами инструментами. Вот лишь несколько примеров взаимосвязей между инструментальными средствами, существующими сегодня:

- вы можете устанавливать связи между требованиями в RequisitePro и вариантами использования, смоделированными в Rational Rose, а также вариантами тестирования, хранящимися в Rational TeamTest;
- DOORS позволяет трассировать требования вплоть до отдельных конструктивных элементов, хранящихся в Rational Rose, Telelogic Tau и других инструментах проектирования;
- RequisitePro и DOORS могут устанавливать связи между отдельными элементами проектного задания в Microsoft Project;
- CaliberRM имеет централизованную структуру коммуникаций, которая позволит вам связать требования с вариантами использования, классами или элементами дизайна (проектирования) процессов, хранящимися в TogetherSoft Control Center, с исходным кодом, хранящемся в Borland's StarTeam, и с тестовыми элементами, хранящимися в Mercury Interactive'sTestDirector. Вы сможете получать доступ к этим взаимосвязанным элементам непосредственно из требований, хранящихся в базе данных CaliberRM.

Оценивая программные средства, подумайте, как вы сможете воспользоваться преимуществом интеграции продуктов при составлении требований, тестировании, трассировании проекта и других процессах. Например, подумайте, как опубликовать основной набор требований в инструменте для управления версиями продукта и определить связи трассирования между функциональными требованиями и конкретными элементами дизайна или кода.

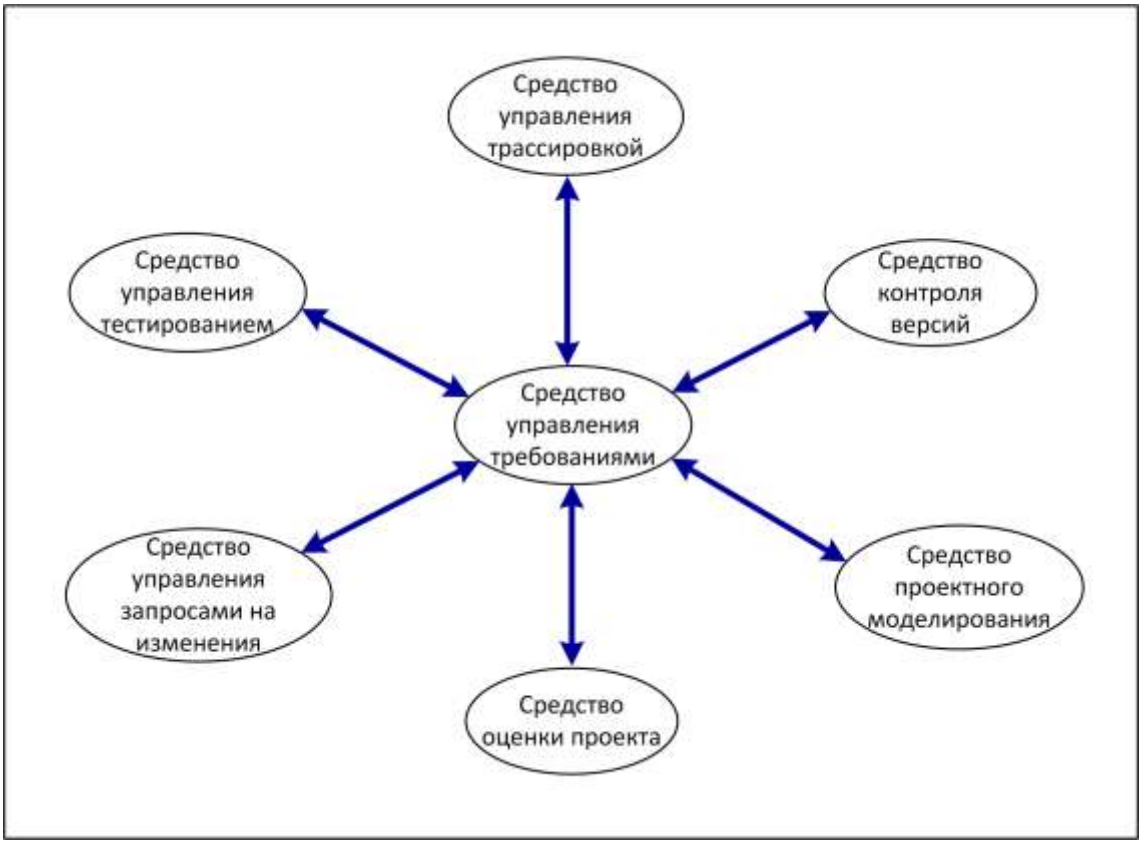


Рис. 21-1. Инструментальные средства управления требованиями интегрируются с другими видами программных средств

Реализация автоматизации управления требованиями

Любой из этих продуктов позволяет управлять требованиями на более тонком и производительном уровне. Тем не менее, старание пользователей инструментальных средств остается критическим фактором успеха. Усердные, дисциплинированные и знающие люди добиваются успехов, даже работая с посредственным инструментарием, тогда как самые лучшие средства не оправдают даже собственной стоимости в руках пользователей, не обладающих должной мотивацией или подготовкой. Не выписывайте чек на оплату средства управления требованиями, если вы не готовы обучать пользователей [вам придется учесть ***кривую обучения*** (learning curve) и затраты времени]. Вы не можете ожидать немедленных результатов, поэтому не рассчитывайте, что проект сразу станет успешным из-за применения инструментального средства. Прежде наберитесь опыта, работая с инструментом над пробными проектами, и лишь затем применяйте его в важном проекте.

Дополнительная информация

В главе 22 описано, как кривая обучения связана с применением новых инструментальных средств и методик.

Выбор инструментального средства

Выбирайте инструментальное средство, учитывая такие параметры, как платформа, цена, режим доступа и способ хранения данных для требований — документальный или в базе данных, которые лучше всего подходит вашей среде разработки и культуре. Некоторые компании нанимают для оценки инструментальных средств консультантов, которые могут оценить все

нужды компании и порекомендовать доступный продукт. Если вы проводите оценку самостоятельно, перечисленные далее рекомендации помогут вам сделать правильный выбор.

1.Во-первых, определите требования организации к средству управления требованиями. Установите, какие возможности для вас наиболее существенны, с какими еще средствами вам хотелось бы интегрировать продукт, насколько важен для вас удаленный доступ к данным через Интернет. Решите, хотите ли вы использовать для хранения некоторой информации о требованиях текстовые документы или предпочтете базу данных.

2.Перечислите 10-15 факторов, которые влияют на ваш выбор. Учтите такие субъективные категории, как возможности индивидуальной настройки, эффективность продукта и интерфейса пользователя. Стоимость, конечно же, важна, но прежде оценивайте инструментальные средства, не оглядываясь на их цену.

3.Оцените факторы выбора, перечисленные в пункте 2, в очках (всего 100 очков), назначая большее значение тем факторам, которые вы считаете более важными.

4.Соберите текущую информацию о доступных инструментальных средствах управления требованиями и сравните их по каждому из определенных вами факторов. С оценкой субъективных факторов придется подождать до тех пор, пока вам не представится возможность непосредственно поработать с каждым средством, Демонстрация продукта производителем может прояснить некоторые вещи, но скорее всего, вам покажут наиболее сильные стороны инструмента. Демонстрация не заменит самостоятельную работу с продуктом в течение нескольких часов.

5.Подсчитайте общую сумму отдельно для каждого инструмента, сложив очки, которые вы определили для каждого пункта, и вы поймете, какие продукты подходят вам лучше всего.

6.Попросите других пользователей составить отчеты об использовании каждого продукта, например, разместив запросы на форумах в Интернете, чтобы дополнить вашу собственную оценку, а также запросите у поставщика литературу, демо-версию продукта и данные о цене.

7.Попросите производителей прислать вам для оценки версии инструментальные средства, занявшие верхние места в вашем рейтинге. Определите, как будете проводить оценку до того, как устанавливать их на компьютер, чтобы быть уверенным, что вы получите необходимую для принятия решения информацию.

8.Оцените инструментальные средства, применяя их в реальном проекте, а не только в обучающем проекте, поставляемом с продуктом. Завершив оценку, при необходимости уточните параметры оценки и выясните, какой из продуктов теперь лидирует.

9.Примите решение, учитывая как количество очков, набранных каждым продуктом, стоимость лицензий и расходов на сопровождение, так и мнение других пользователей и субъективные впечатления членов вашей команды от каждого продукта..

Изменение культуры работы

Купить инструментальное средство легко; изменить культуру и организацию работы, чтобы принять это средство и извлечь из него максимальную пользу, гораздо сложнее. Большинство организаций уже привыкли (и чувствуют себя при этом весьма комфортно!) хранить требований в форматах текстовых редакторов. Для перехода к оперативным приемам работы с требованиями необходим новый образ мышления. Инструментальное средство открывает доступ к требованиям в базе данных любому участнику проекта, имеющему соответствующие разрешения. Некоторые воспринимают это как уменьшение контроля над требованиями, которым они раньше обладали, над процессом создания требований или над тем и другим. Другие предпочитают не показывать миру неполные или несовершенные спецификации требований к ПО, ведь содержание базы данных доступно всем. Скрывая требования до того, как они «готовы», вы упустите возможность показать их многим потенциальным критикам — а это отличный способ выявления возможных проблем.

Нет большого смысла применять инструментальное средство управления требованиями, если вы не воспользуетесь его возможностями. Я знаю команду, которая, работая над проектом, прилежно сохраняла все требования в таком средстве, но не определяла для этих требований никаких атрибутов или связей трассирования. Не предоставляли они и оперативного доступа заинтересованным в проекте лицам. Тот факт, что требования хранились в иной форме, не дал

существенных преимуществ, хотя было затрачено много сил для ввода, требований в программу. Другая команда хранила сотни требований в инструментальном средстве и установила много связей трассирования. Но единственное для чего использовались эти данные, была генерация массы предназначенных для печати отчетов трассирования, которые потом предполагалось вручную проверять на ошибки. Никто эти отчеты не изучал, и никто не считал базу данных заслуживающим доверие хранилищем требований. Ни одна из этих организаций не получила полной выгоды от существенных инвестиций времени и денег в инструментальные средства управления требованиями.

Учитывайте следующие вопросы корпоративной культуры и организации процесса, если хотите получить максимальную выгоду от вложений в коммерческие инструментальные средства управления требованиями:

- даже не пробуйте использовать инструментальное средство, пока ваша организация не сможет создать приемлемую спецификацию требований к ПО на бумаге. Если ваша основная проблема связана со сбором и записью ясных и качественных требования, эти средства вам не помогут;
- не пытайтесь вносить требования непосредственно в инструментальное средство, когда только начинаете проводить семинары для выявления требований. Однако, по мере того как требования начнут обретать устойчивые формы, хранение их в программе откроет доступ к ним участникам семинаров для уточнения;
- используйте инструмент как средство налаживания связей между заинтересованными в проекте лицами, которые находятся в различных регионах. Установите доступ и измените привилегии различным людям, чтобы позволить им ввод данных в требования, не давая при этом им права изменять всю базу данных;
- тщательно продумывайте, какие типы требований вы определяете. Не считайте каждый раздел вашей нынешней спецификации требований отдельным типом требований, но и не помещайте все содержимое спецификации в единственный тип требований. Инструментальные средства позволяют создавать различные атрибуты для каждого определяемого типа требований, так что выбор соответствующих атрибутов поможет вам решить, сколько различных типов требований определять;
- назначьте ответственного для каждого типа требований, который будет отвечать за управление информацией этого типа в базе данных;
- используйте деловую, а не ИТ-терминологию, определяя новые поля данных или атрибуты требований;
- не определяйте связей трассирования, пока требования не обретут устойчивые формы. В противном случае вам придется исправлять массу связей по мере изменения требований;
- для ускорения перехода от документального способа хранения данных к использованию набора инструментальных средств, назначьте дату, после которой база данных инструментального средства будет считаться окончательным хранилищем требований проекта. После этой даты все требования, содержащиеся в формате текстовых редакторов, не будут считаться документами, имеющими силу;
- не ожидайте фиксации требований на ранних стадиях проекта, выработайте привычку создавать основную версию требований для каждого конкретного выпуска. При необходимости динамически переносите требования из основной версии одного выпуска в основную версию другого выпуска.

Дополнительная информация

В главе 18 говорилось об использовании атрибутов требований для управления требованиями, предназначенными для различных выпусков.

По мере того, как инструментальное средство управления требованиями все больше становится частью корпоративной культуры, участники проекта начнут воспринимать требования как необходимый элемент работы, как, например, кодирование. Команда будет открывать способы применения инструментального средства для ускорения процесса документирования требований, их передачи и управления изменениями в них. Вместе с тем помните: даже самое лучшее инструментальное средство не сможет восполнить пробелы неэффективного процесса разработки требований. Программа не поможет вам оценить масштабы своего проекта, определить пользователей, поговорить с нужными пользователями или собрать корректные требования. И не важно, насколько хорошо вы будете управлять плохими требованиями.

Как заставить инструментальные средства работать

Для более плавного перехода к работе с новыми средствами назначите «защитника» инструментального средства, энтузиаста, который сам изучит инструмент вдоль и поперек, будет учить других пользователей и следить, чтобы его применяли как нужно. Начните с пробного применения инструментального средства в не очень важном проекте. Это поможет понять, сколько усилий необходимо для управления инструментальным средством и его поддержки. Первый «защитник» инструментального средства будет управлять его использованием в пробном проекте, а затем обучать и наставлять других по мере применения этого средства в других проектах. Несомненно члены вашей команды сообразительны, но лучше обучить их, чем ждать, когда они сами освоят инструмент. Они, конечно же, додумаются, как осуществлять основные операции, но полный набор возможностей инструмента и их эффективная эксплуатация останутся для них тайной за семью печатями.

Учитывайте, что загрузка требований проекта в базу данных, определение атрибутов и связей трассирования, своевременное обновление содержимого базы данных, определение доступа групп и их привилегий, а также подготовка пользователей потребуют усилий. Руководство должно выделить ресурсы, необходимые для этих операций. Доведите до сведения всех сотрудников вашей фирмы просьбу действительно использовать выбранный продукт, иначе он просто превратится в дорогое украшение, которое лежит на полке.

Что теперь?

- Проанализируйте недостатки вашей существующей организации управления требованиями, чтобы оценить инструментальное средство и понять, оправдает ли оно вложения. Важно, чтобы вы понимали причины существующих проблем; не нужно слепо верить, что инструмент решит их.
- До сравнения предлагаемых продуктов оцените готовность вашей команды к использованию инструмента. Вспомните предыдущие попытки включить новые инструментальные средства в процесс разработки ПО. Проанализируйте их успех или неудачу, чтобы настроить себя на успех в этот раз.

Если вы не забыли, что инструментальное средство не может исправить недостатки процесса, то скорее всего согласитесь с тем, что коммерческие средства управления требованиями помогают вам управлять требованиями к ПО. Если вы заставите базу данных с требованиями работать на себя, то вряд ли когда-нибудь вернетесь к простой бумаге.