Глава 27

Критерии качества требований к программному обеспечению

Основные положения

- Основным критерием качества является сам факт наличия набора требований; в свою очередь, эти требования также должны удовлетворять критериям качества.
- Для того чтобы удостовериться в качестве требований, модели прецедентов, спецификаций прецедентов и акторов, можно использовать контрольные перечни.
- Высококачественный пакет Modern SRS Package должен иметь хорошее оглавление, индекс, глоссарий, а также должен содержать историю исправлений.

Качество, как и мастерство в искусстве, трудно измерить. ("Я узнаю настоящее искусство, когда увижу его.") Тем не менее нам нужно найти способ измерять и повышать качество наших спецификаций. Очевидно, что одной из мер качества является получение в результате завершения проекта хорошего продукта. Но от такой меры проку немного, так как помимо требований существует много других факторов, влияющих на результат.

Мы предлагаем оценивать качество спецификации в целом, рассматривая следующие основные элементы.

- Качество каждой отдельной спецификации
- Качество применяемых методов (например, качество спецификаций прецедентов, акторов и т.п.)
- Качество пакета, объединяющего все отдельные спецификации

Начнем с рассмотрения качества отдельных спецификаций. Чтобы нечто измерить, естественно, необходимо иметь способ проведения измерений.

Девять показателей качества

Мы уже обсуждали, что собой представляют "хорошие" требования: они удовлетворяют предложенному определению (23) и не содержат описания деталей проектирования и реализации, а также вопросов процесса программирования или управления проек-

том. Но как отличить качественный набор требований от некачественного? Руководствуясь стандартом IEEE 830, который выделяет восемь "критериев качества" для оценки SRS, мы добавим еще один и объясним, как он поможет нам разработать качественный набор требований. Итак, высококачественный пакет Modern SRS Package должен быть

- корректным;
- недвусмысленным;
- полным;
- непротиворечивым;
- упорядоченным по важности и стабильности;
- поддающимся проверке;
- модифицируемым;
- трассируемым;
- понимаемым.

Мы добавили девятый критерий, *возможность понимания*, так как твердо верим, что важнейшим залогом успеха проекта является *общение* его участников.

Обеспечение качества набора требований второстепенно по сравнению с задачей создания этого набора. Тем не менее, обсудим более подробно каждый из этих девяти элементов, так как они придают дополнительную глубину процессу разработки требований, а также позволяют нам лучше понять природу "хороших" требований.

Корректные требования

Дэвис (Davis, 1993) предложил следующую формулировку и ее иллюстрацию (рис. 27.1): "SRS (набор требований к программному обеспечению) является корректным тогда и только тогда, когда каждое требование, сформулированное в нем, представляет нечто, требуемое от создаваемой системы".

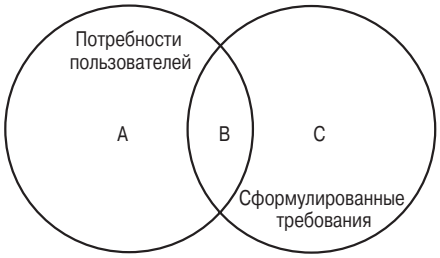


Рис. 27.1. Множества потребностей и требований

Если левый круг (область A) представляет множество потребностей пользователя, а правый (область C) — требования, корректные требования будут находиться в области пересечения кругов, области B.

Просто записывая некую информацию в документ, нельзя гарантировать, что она корректна; такую гарантию не сможет обеспечить и применение любого средства автоматизированного проектирования. Если истинное требование пользователя для системы расчета заработной платы состоит в том, чтобы ставка всех служащих была автоматиче-

ски повышена на 5 процентов, а команда разработчиков по невнимательности создает требование, предусматривающее 10-процентное повышение ставки, это, безусловно, некорректно. Но проверить это можно только с помощью просмотра пользователем.

Это явление нельзя назвать новым и неизвестным; с ним команды проектировщиков сталкивались с начала работы над самыми первыми проектами программных систем, равно как и при работе над другими проектами. Но в программных проектах часто встречается ситуация, когда опускается информация из области A и включается нежелательная информация из области C. Информация из области C может представлять собой детали проектирования/реализации, о чем мы предупреждали ранее, но это могут быть также требования, которые пользователь никогда не высказывал. Иногда эту информацию включают энтузиасты из числа персонала, занимающегося маркетингом или техническими аспектами, которые думают: "Мы уверены, что пользователь придет в восторг от этой функции, как только увидит ее". (Мы вновь вернемся к теме неправомерных требований в части 6.)

Недвусмысленные требования

Требование является недвусмысленным *тогда и только тогда, когда его можно однозначно интерпретировать* (IEEE 830-1993, § 4.3.2, 1994). Хотя главным свойством любого требования по праву считается корректность, неоднозначность зачастую представляет собой более сложную проблему. Если формулировка требований может по-разному интерпретироваться разработчиками, пользователями и другими участниками проекта, вполне может оказаться, что построенная система будет полностью отличаться от того, что представлял себе пользователь. Эта проблема всегда может возникнуть, если требования пишутся на естественном языке, так как члены организации, относящиеся к разным группам, настолько привыкли к своей интерпретации слова или фразы, что им никогда и в голову не придет, что другие могут интерпретировать это слово иначе (эту проблему мы обсуждали в главе 26).

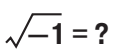
Полнота набора требований

Набор требований является полным *тогда и только тогда, когда он описывает все важные требования, интересующие пользователя, в том числе требования, связанные с функциональными возможностями, производительностью, ограничениями проектирования, атрибутами или внешними интерфейсами* (IEEE 830-1993, § 4.3.3, 1994). Полный набор требований должен также задавать требуемый ответ программы на всевозможные классы ввода — как правильные, так и неправильные — во всевозможных ситуациях. Помимо этого, он должен содержать полные ссылки и подписи всех рисунков, таблиц и диаграмм набора требований, а также определения всех терминов и единиц измерения.

Гарантия полноты

О некоторых аспектах полноты может судить любой компетентный рецензент, который критически оценивает пакет программных требований, чтобы удостовериться, что все рисунки, таблицы и диаграммы снабжены соответствующими ссылками и подписями. Кое-что может оценить даже сам разработчик, не имея специальных знаний о приложении. Например, если в требованиях указано: *система должна принимать вводимое пользователем отдельное число и возвращать квадратный корень из него с точностью до трех знаков после запятой*, возникает очевидный вопрос: *что будет, если пользователь попытается ввести отрицательное число?*

Вообще-то, ничего страшного в попытке вычислить квадратный корень из отрицательного числа нет, если в приложении имеет смысл возвращение в качестве ответа системы мнимого числа. Но в таком случае уже требуется понимание проблемной области, чтобы отличить правильный и неправильный ввод. Эта проблема особенно часто встречается при задании верхнего и нижнего пределов вводимых числовых параметров, длины символьных строк и т.п. Поскольку эти детали зачастую игнорируются в требованиях, разработчику приходится осознанно или неосознанно принимать решение, и в результате получается система, которая отказывается принимать имя клиента, состоящее из более чем 25 символов, или выдает причудливые результаты при ошибочном вводе пользователя.



Просмотр возможных классов вводов с целью удостовериться в том, что полный набор требований надлежащим образом описывает поведение системы при правильных и неправильных вводах, — это то, о чем знает каждый разработчик, хотя мы до сих пор делаем ошибки в написании таких требований. Именно для пользователей типично игнорировать эту область при обсуждении: "Почему нормальный человек будет пытаться ввести отрицательное число, когда система спрашивает о возрасте?". Опытный разработчик знает, что это может случиться из-за простой опечатки или потому, что пользователь умышленно пытается "повредить" систему либо по другим неясным причинам.

Полнота нефункциональных требований

Чаще других пропускают аспекты производительности и ограничения проектирования, а также предположения о внешних интерфейсах с другими системами. Мы советуем (следуя нашим указаниям, которые мы предлагали при обсуждении практичности, надежности, производительности и возможности сопровождения) создать простой контрольный перечень (checklist), содержащий вопросы, которые необходимо задать при выявлении ограничений проектирования. При этом разработчики и пользователи, по крайней мере, могут быть уверены в том, что они задали соответствующие вопросы при создании требований. Пользователь-новичок вновь может выразить недовольство: "Конечно же, я хочу, чтобы система имела хорошие характеристики производительности, это настолько очевидно, что я не понимаю, зачем специально указывать это". Опытный разработчик знает, насколько важно указать требования производительности в виде максимального и среднего времени ответа или в виде утверждения: "Время ответа для 90 процентов всех транзакций будет составлять менее 3 секунд".

Полнота функциональных требований

Вопросы полноты функциональных требований более сложные. Не будучи экспертом в области приложения, техническому разработчику очень трудно узнать, не пропущена ли важная часть функциональных возможностей. В конце концов, поскольку все функциональные возможности новые, откуда вы можете знать, сколько их еще должно быть?

Иногда эти возможности настолько привычны и "очевидны", что пользователь даже не осознает, что они есть. ("Конечно, мы запускаем систему расчета заработной платы на день раньше, если день выплаты приходится на праздник. Мы всегда так делаем! Какие другие варианты вы можете себе представить?")

В этом случае может помочь метод прецедентов.

Достижение полноты с помощьюпрототипирования

Раскадровка, проверка требований и создание прототипов системы при использовании итеративного подхода к разработке помогут справиться с большей частью этих проблем. Чем ближе мы подходим к реальному использованию и чем больший опыт работы с внедряемой системой приобретают наши пользователи, тем выше вероятность заметить проблемы в нашем определении.

Но даже тогда команда разработчиков должна идти на шаг вперед в своем анализе и задавать всевозможные вопросы "что, если…" с целью гарантировать полноту требований. При этом следует обратить внимание на граничные условия, исключения и неординарные события.

Например, иногда в описании функциональных возможностей представлены ситуации столь редкие, что они никогда не возникали в процессе деловой жизни пользователя; никто и не думает составлять требования для подобной ситуации. Пользователь системы начисления зарплаты может считать, что требуемое поведение системы "очевидно" в случае, если день выплаты приходится на выходной. Но что если национальный праздник, праздник штата и городской праздник придутся на три последовательных рабочих дня? "Такого не бывает", — может возразить пользователь, но разработчик в состоянии продемонстрировать, что такое может произойти в течение последующих 5 лет.

Эти вопросы не так уж нереальны, как может показаться. Суматоха вокруг "проблемы 2000" наглядно иллюстрирует последствия близоруких решений, основанных на "обоснованных" ожиданиях, касающихся будущих событий.[1]

Непротиворечивость набора требований

Множество требований является внутренне непротиворечивым *тогда и только тогда, когда ни одно его подмножество, состоящее из отдельных требований, не противоречит другим подмножествам* (IEEE 830-1993, § 4.3.4.1, 1994). Конфликты могут иметь различную форму и проявляться на различных уровнях детализации; если набор требований был написан достаточно формально и поддерживается соответствующими автоматическими средствами, конфликт иногда удается обнаружить посредством механического анализа. Но, скорее всего, разработчикам вместе с другими участниками проекта придется провести проверку множества требований вручную, чтобы удалить все потенциальные конфликты.

Иногда конфликты бывают явными и очевидными. Например, одна часть требований гласит: *когда происходит X, следует выполнять действие P*, в то время как другая часть утверждает: *при возникновении X выполнять Q*. Порой неясно, является ли проблема конфликтом или следствием неоднозначности. Например, часть требований в системе расчета заработной платы может выглядеть так: *все служащие, достигшие 65 лет и более, в конце календарного года должны получить поощрение $1000*, а другая часть гласит: *все сотрудни-*

[1] Известно, что в свое время были приняты неблагоразумные решения, приведшие к возникновению "проблемы 2000". Существует и множество других примеров. Например, при подготовке полета космического корабля Джеминай (Gemini) была допущена ошибка в программе бортового компьютера, вызванная сознательным пренебрежением определенными законами физики с целью повышения эффективности программы. Принятое тогда решение привело к тому, что место приземления на несколько сотен миль отличалось от расчетного.

ки, имеющие стаж работы 10 лет и более, в конце календарного года должны получить поощрение в сумме $500. Как в этом случае поступить с работниками, для которых выполнены оба условия?

В данном случае прототипы мало чем могут нам помочь. Хотя они весьма успешно позволяют выявить пропущенные функции и очень эффективны при проверке требований пользователя, касающихся деталей ввода-вывода, редко бывает, чтобы пользователь или специалисты по тестированию или обеспечению качества работали с прототипом настолько тщательно, чтобы выявить наиболее скрытые ошибки-конфликты. Их необходимо выявлять с помощью тщательной выполняемой вручную ревизии и анализа полного набора требований, опираясь на профессиональные навыки команды разработчиков и имеющиеся в наличии автоматизированные средства.

Упорядочение требований по их важности и стабильности

В высококачественном наборе требований разработчики, клиенты и другие заинтересованные лица упорядочивают отдельные требования по их важности для клиента и стабильности (IEEE 830-1993, § 4.3.5, 1994). Этот процесс упорядочения особенно важен для управления масштабом. Если ресурсы недостаточны, чтобы в пределах выделенного времени и бюджета реализовать все требования, очень полезно знать, какие требования являются не столь уж обязательными, а какие пользователь считает критическими.

Можно присвоить дополнительные атрибуты каждому требованию, как мы делали при описании требований в документе-концепции; особенно полезны стоимость, риск, сложность и т.п. Но многие из них, по всей видимости, будут зависеть от оценки стратегий реализации. Например, посмотрев на требование, разработчик может сказать: “Хм, я думаю, что реализовать это требование будет действительно очень сложно, я даже не уверен, что мы знаем, как вообще это сделать”. Хотя эта информация и важна для управления масштабом и принятия решений об очередности реализации, она, как правило, описывается на более высоком уровне абстракции, представленном документом-концепцией. По причинам, которые мы уже обсуждали, эти элементы обычно н*е следует* включать в набор требований.

На данном этапе лучше использовать атрибуты “важности” и “стабильности”, которые более тесно связаны с мировосприятием пользователя. Пользователь может сказать: “Это требование не очень стабильно, поскольку мы ожидаем, что в будущем месяце изменятся государственные инструкции, влияющие на него. Но с другой стороны, оно очень важно для нас, так как влияет на нашу конкурентоспособность”. До того, как команда разработчиков начнет разрабатывать стратегии, основанные на технологических возможностях, полезно упорядочить требования, например, с помощью состоящей из двух столбцов таблицы.

Требования, упорядоченные по их важности	**Требования, упорядоченные по их стабильности**
SR103	SR172
SR172	SR103
SR192	SR063
SR071	SR071
SR063	SR192

Обладая этой информацией (в случае равенства других факторов), благоразумный менеджер разработки выделит пропорционально бо́льшую часть ресурсов на SR103 и SR172 и, скорее всего, вычеркнет SR071, так как оно не столь важное и относительно непостоянное по своей природе.

Проверяемые требования

> Требования должны быть верифицируемыми (или тестируемыми).

Требование в целом является верифицируемым *тогда и только тогда, когда каждое из составляющих его элементарных требований является верифицируемым. Элементарное требование считается верифицируемым тогда и только тогда, когда существует конечный финансово эффективный процесс, с помощью которого человек или машина могут определить, что разработанная программная система действительно удовлетворяет данному требованию* (IEEE 830-1993, § 4.3.6, 1994). Если сказать проще, *практическая* задача состоит в таком определении требований, чтобы можно было впоследствии протестировать их и выяснить, действительно ли они выполняются.

Вряд ли можно предложить строго научное доказательство того, что каждое требование является верифицируемым. В большинстве случаев это и не нужно. Создание соответствующих тестовых примеров и процедур для проведения верификации разработанной программы является задачей персонала, занимающегося тестированием и обеспечением качества. Конечно, чтобы иметь возможность сделать это, они нуждаются в хорошо определенных и недвусмысленных требованиях. На совещании, посвященном проверке, типичной является картина, когда каждый из участников обращается к специалистам по тестированию и спрашивает: "Вы уверены, что можете создать тестовый сценарий для проверки того, что данное требование выполнено?

Ниже приводятся примеры требований и типичные высказывания разработчиков и/или специалистов по тестированию о возможности их верификации.

- *Система должна поддерживать до 1000 пользователей одновременно.* "Это зависит от того, что разрешено делать этим пользователям после регистрации. Если пользователи имеют *неограниченные* возможности и теоретически могут ввести транзакцию, которая приведет к тому, что прикладная программа будет последовательно просматривать каждую запись базы данных, очень сложно проверить, сможет ли система работать с 1000 пользователями; существует ничтожная (но не нулевая) вероятность, что все эти пользователи одновременно захотят запустить такие транзакции. Но если пользователи ограничены в выборе запускаемых транзакций и мы сможем определить, какая из этих транзакций является наиболее трудоемкой, можно будет проверить, как удовлетворяется это требование (с разумной степенью достоверности), хотя нам придется использовать наше средство контроля загрузки для имитации 1000 активных терминалов.
- *Система должна отвечать на произвольный запрос в течение 500 миллисекунд.* "Все зависит от того, что подразумевается под словом *произвольный*. Если число возможных запросов конечно и нам удастся выявить наиболее сложные из них, мы сможем проверить поведение системы.

- *Цифры на экране, показывающем время, должны хорошо выглядеть.* "Даже не думайте об этом. Красота – вопрос вкуса."

- *Система должна быть дружественной пользователю.* "Это еще хуже, чем *приятная форма!* Если не будут тщательно определены условия и детали, *дружественность пользователю* является просто приглашением для введения аргументов."
- *Система должна экспортировать данные для просмотра (view data) в формате, в котором в качестве разделителя используется запятая.* "Хотелось бы уточнить некоторые детали; например, что будет, если эти данные представляют собой пустое множество? Но в принципе, мы можем проверить, будет ли система вести себя нужным образом в данном вопросе."

Верификация и проверка правильности являются важными вопросами при разработке высококачественной программы. Мы вернемся к этой теме в части 6.

Модифицируемый набор требований

Множество требований является модифицируемым *тогда и только тогда, когда его структура и стиль таковы, что любое изменение требований можно произвести просто, полно и согласованно, не нарушая существующей структуры и стиля всего множества* (IEEE 830-1993, § 4.3.7, 1994). Для этого требуется, чтобы пакет имел минимальную избыточность и был хорошо организован, с соответствующим содержанием, индексом и возможностью перекрестных ссылок. Это не всегда означает, что пакет ведется и поддерживается с помощью некоего автоматического средства, но в больших системах, которые могут иметь тысячи требований, использование подобных средств становится практически необходимым.

Требования *будут* модифицироваться, нравится это кому-то или нет; альтернатива — это "замороженный" пакет требований, что равносильно его отсутствию и, соответственно, "провалу" проекта. Если требование (или содержащий его пакет) немодифицируемо, оно фактически становится недействительным через несколько недель или месяцев, так как команда постепенно откажется от своих усилий по его изменению и поддержанию его актуальности.

Каждому администратору программного продукта хочется думать, что его множество требований является модифицируемым, и каждый производитель вспомогательных программ хвастается, что одним из главных достоинств его средства является то, что оно действительно обеспечивает модифицируемость. Все это звучит красиво, но нужно его опробовать, чтобы увидеть, работает ли оно. И это нужно делать для того же масштаба и уровня сложности, что и в будущем проекте.

Трассируемые требования

Требование в целом является трассируемым *тогда и только тогда, когда ясно происхождение каждого из составляющих его элементарных требований и существует механизм, который делает возможным обращение к этому требованию при дальнейших действиях по разработке* (IEEE 830-1993, § 4.3.8, 1994). На практике это обычно означает, что каждое требование имеет уникальный номер или идентификатор. Иногда можно использовать ключевое слово, например "должна" (shall), чтобы выделить требование и отличить его от других не столь важных утверждений, комментариев и т.п., которые также могут содержаться в набо-

ре требований. При использовании автоматического средства работы с требованиями, идентификация требований может осуществляться системой автоматически.

В пределах одного проекта или, возможно, одного пакета будет необходимо трассировать одни компоненты к другим. Например, некоторые компоненты будут зависеть от других; если одни меняются, это повлияет и на другие. Некоторые утверждения требований могут уточняться "дочерними" требованиями (субтребованиями), для которых возможность трассировки является очевидным свойством. Нам необходима возможность *обратной* трассировки (backward traceability) — от текущих стадий к предыдущим стадиям определения или разработки, в частности, к документу-концепции, который мы обсуждали в части 3. Например, из табл. 23.1 видно, что все требования SR63.1, SR63.2, SR63.3 можно трассировать к Функции 63 документа-концепции (Vision document). Это очень существенно, если нам понадобится добавить или исключить некоторые функции; это также существенно, если возникают трудности с определенными требованиями и необходимо вновь согласовывать с пользователем сроки или бюджет для затронутой этим процессом функции. Также необходима возможность *прямой* трассировки (forward traceability) — от текущего требования ко всем подчиненным ему требованиям, независимо от того, какие контейнеры (документы проектирования, блок-схемы, программный код, тестовые примеры и т.п.)порождаются данным контейнером.

Возможность трассировки имеет огромное значение. Разработчики могут использовать ее как для достижения лучшего понимания проекта, так и для обеспечения более высокой степени уверенности, что все требования выполняются данной реализацией. Например, мы использовали полную трассировку, чтобы соединить все элементы одного нашего медицинского проекта (среди которых были элементы документа-концепции, элементы SRS, тестирования, кодирования и ревизии проекта), которые возникали на протяжении жизни проекта. Когда все эти элементы взаимосвязаны, разработчики гораздо лучше подготовлены к управлению взаимодействиями между ними.

Трассировка также позволяет команде решать вопросы "что, если…". ("Что, если мы прямо сейчас изменим это требование? Повлияет ли это на разработку программы, и, если да, то на какие элементы? Придется ли нам пересматривать планы тестирования, и, если да, то какие?")

В том же проекте мы создавали *матрицы трассировки*, требуемые FDA для того, чтобы удостовериться, что продукт соответствует ее собственным требованиям. Матрицы трассировки — бесценный способ "проверить" действия по разработке и убедиться, что вы делаете все необходимое (и не делаете того, что делать не следует). Мы проведем масштабное исследование возможностей трассировки в части 4.

Понимаемые требования

Множество требований является *понимаемым*, если пользователи и разработчики способны прийти к полному согласию относительно отдельных требований и общих функциональных возможностей, подразумеваемых данным множеством. В документах, описанных в предыдущих главах данной книги, основное внимание уделяется общим описаниям и функциям системы, и их понять, как правило, не сложно. Но по мере уточнения определения системы, т.е. при разработке детальных требований, элементы становятся все более конкретизированными и детализированными и возникает искушение использовать более специальные термины. Таким образом, человек, который пишет требования, должен знать терминологию обеих сторон-участниц. Важно также, чтобы те,

кто пользуется набором требований, могли понять поведение системы в целом. Этого можно добиться с помощью раскадровок, сценариев или иллюстративных прецедентов, которые показывают, как предполагается использовать систему в ее операционной среде.

Показатели качества для модели прецедентов

Замечание. В этом разделе обсуждается широкий спектр вопросов, связанных с прецедентами. Иногда элемент контрольного перечня будет содержать ссылки на специфические элементы прецедента, которые не обсуждались в данной книге, так как они не очень важны для управления требованиями. Чтобы изучить эти вопросы, можно обратиться к соответствующей литературе, посвященной прецедентам. Мы рекомендуем две книги: Буч (Booch, 1999), а также Джейкобсон, Буч и Рамбо (Jacobson, Booch, Rumbaugh, 1999).

- Все ли прецеденты найдены? Выявленные прецеденты должны иметь возможность осуществить все варианты поведения системы; если это не так, значит, некоторые прецеденты пропущены.
- Все ли функциональные требования описываются прецедентами? Если вы намеренно отложили некоторые требования, чтобы заняться ими при объектном моделировании (например, нефункциональные требования), это необходимо где-то отразить. Если подобное требование затрагивает конкретный прецедент, это следует отметить в специальном разделе данного прецедента (Специальные требования, Special Requirements).
- Не содержит ли модель прецедентов ненужное поведение, т.е. не представляет ли больше функций, чем указано в требованиях?
- Действительно ли в модели необходимы все выявленные связи включения, наследования и генерализации? Если это не так, они могут быть избыточными, и их следует удалить.
- Зависят ли связи модели друг от друга? Важно, чтобы этого не происходило, поэтому нужно проверить это.
- Правильно ли произведено деление модели на пакеты прецедентов? Стала ли модель в результате проще и удобнее для восприятия и сопровождения?
- Можете ли вы, изучив модель прецедентов, составить четкое представление о функциях системы и о том, как они связаны?
- Содержит ли введение (Introduction) к модели всю необходимую информацию?
- Содержит ли общее описание (Survey Description) модели прецедентов всю необходимую информацию; например, представлены ли там наиболее типичные последовательности прецедентов?

Спецификации прецедентов

- Каждый ли прецедент имеет хотя бы один актор? В противном случае что-то неправильно; прецедент, который не взаимодействует ни с одним актором, следует удалить.

- Все ли прецеденты независимы друг от друга? Если два прецедента всегда активизируются в одной и той же последовательности, возможно, их удастся объединить в один прецедент.
- Есть ли прецеденты с очень похожим поведением или похожими потоками событий? Если да и вы хотите, чтобы их поведение оставалось таким же и в будущем, нужно объединить их в единый прецедент. Тогда в будущем будет проще производить необходимые изменения. *Замечание.* Если вы принимаете решение о слиянии прецедентов, необходимо проинформировать об этом пользователей, так как это может оказать влияние на тех из них, кто взаимодействует с объединяемыми прецедентами.
- Не получилось ли так, что часть потока событий уже моделировалась в качестве другого прецедента? Если да, то можно в новом прецеденте использовать старый.
- Не является ли часть потока событий частью другого прецедента? Если да, то следует выделить этот субпоток и предоставить возможность рассматриваемым прецедентам использовать его. *Замечание.* Необходимо проинформировать пользователей при принятии решения "повторно использовать" субпоток, так как это может повлиять на тех из них, кто использует существующий прецедент.
- Будет ли поток событий одного прецедента включаться в поток событий другого? Если да, это следует моделировать с помощью отношений наследования прецедентов. (Мы не обсуждали этот вопрос, так как он не был столь важен для общей концепции прецедентов.)
- Имеют ли прецеденты уникальные, понятные и содержательные имена, чтобы не перепутать их на последующих этапах? Если нет, имена следует изменить.
- Понимают ли пользователи и клиенты имена и описания прецедентов? Каждое имя должно описывать поведение, поддерживаемое соответствующим прецедентом.
- Удовлетворяет ли прецедент все требования, которые очевидным образом регулируют его выполнение? Можно включить любые нефункциональные требования, описанные в других частях пакета Modern SRS Package
- Соответствует ли последовательность взаимодействий актора и прецедента ожиданиям пользователя?
- Понятно ли, как и когда начинается и заканчивается поток событий прецедента?
- Может существовать поведение, которое активизируется только в случае невыполнения некоторого условия. Есть ли описание того, что произойдет, если данное условие не выполняется?
- Нет ли слишком сложных прецедентов? Если вы хотите, чтобы модель прецедентов можно было легко понять, нужно "расщепить" сложные прецеденты.
- Содержит ли прецедент раздельные потоки событий? Если это так, лучше разделить его на два или более отдельных прецедентов. Прецедент, содержащий независимые потоки событий, очень сложно понимать и обслуживать.
- Тщательно ли смоделирован субпоток событий в прецеденте?
- Ясно ли, кто хочет выполнять прецедент? Понятно ли назначение прецедента?
- Понятны ли взаимодействия акторами и обмен информацией?
- Передает ли краткое описание истинную природу прецедента?

Акторы прецедента

- Все ли акторы найдены? Иными словами, все ли роли в окружении системы учтены и смоделированы? Несмотря на проводимые проверки, нельзя быть уверенным в этом до тех пор, пока не будут найдены и описаны все прецеденты.
- Каждый актор должен входить, по крайней мере, в один прецедент. Акторы, которые не упоминаются в описаниях прецедентов или не имеют коммуникационных связей с прецедентами, нужно удалить. Но актор, упоминаемый в описании прецедента, скорее всего, имеет с ним некую связь.
- Можете ли вы назвать хотя бы двух человек, которые смогут выполнять действия, как конкретный актор? Если нет, проверьте, не является ли роль, моделируемая актором, частью другой роли. Если это так, следует произвести операцию слияния акторов.
- Есть ли акторы, играющие аналогичные роли по отношению к системе? Если это так, их следует объединить. Коммуникационные ассоциации и описания прецедентов показывают, как взаимосвязаны акторы и система.
- Если два актора играют по отношению к прецеденту одну и ту же роль, необходимо использовать генерализацию для моделирования их общего поведения.
- Если некий актор использует систему с помощью нескольких полностью отличных способов или имеет несколько совершенно различных целей при использовании прецедента, то, вероятно, вы имеете дело с несколькими акторами.
- Акторы должны иметь простые осмысленные имена, которые смогут понять пользователи и клиенты. Важно, чтобы имена акторов соответствовали их ролям. В противном случае их необходимо переименовать.

Критерии качества пакета Modern SRS Package

Помимо индивидуальных критериев качества существуют критерии, применяемые к самому пакету. Эти критерии позволяют удостовериться, что сам пакет является высококачественным и понимаемым.

При работе с документацией сложнее всего найти необходимую информацию. Сколько раз вы говорили: “Я знаю, что требование, касающееся безотказности, находится где-то здесь,” но не могли найти его. Слишком часто мы думаем, что достаточно просто зафиксировать собранные требования. Но это не так.

Мы обнаружили, что Modern SRS Package не просто организовывает и хранит требования; он также *упрощает их использование*. Хороший пакет спецификаций обладает следующими характеристиками.

Хорошо составленное оглавление

Обязательным элементом хорошего SRS является хорошо составленное оглавление (Table of Contents, TOC). Мы убедились, что TOC предоставляет еще больше преимуществ, чем может показаться на первый взгляд. Например, стремление к хорошему TOC подталкивает автора к использованию полезных заголовков, которые затем возникают в оглавлении SRS. Посмотрите на оглавление данной книги, и вы заметите, что заголовки

сами по себе могут служить руководством для читателя. Таким образом, оглавление напоминает сжатую версию пакета.

При наличии современных инструментальных средств непростительно не иметь оглавления. Оглавление создается автоматически; автор может выбрать необходимый уровень детализации и форматирование. Мы считаем, что трех или четырех уровней заголовков вполне достаточно для обеспечения нужного уровня детализации пакета Modern SRS Package. Для того чтобы отделить друг от друга основные элементы оглавления, полезно использовать дополнительные промежутки.

Одной из часто встречающихся проблем является то, что TOC не обновляется так, как это нужно для отражения текущего вида пакета. Следует удостовериться, что в процессе публикации SRS всегда обновляется TOC, чтобы гарантировать правильность разбиения на страницы. Оглавление становится ненужным, если оно не обновляется. Из-за этого читатель начинает беспокоиться, что SRS что-то не так.

К сожалению, оглавление сложно создавать, если необходимо включить в него несколько разнородных документов. Предположим, готовится SRS-пакет, содержащий несколько документов Word, несколько разработанных с помощью другого средства спецификаций прецедентов и несколько выполненных в Excel схем. Невозможно найти средство, которое сможет одновременно работать с подобными входными данными и подготовить хорошее оглавление. В таких случаях можно попытаться создать многотомное оглавление, в котором верхний уровень используется для указания основных групп, таких как документы Word, файлы модели прецедентов, электронные таблицы Excel. Затем можно применять отдельные TOC-средства для создания оглавлений каждой из групп.

Хороший индекс

Индексы являются важной составной частью любого SRS. В отличие от оглавлений, создание индексов — более сложное дело, так как авторы должны определить, по каким элементам будет проводиться индексация. После этого создание индекса не представляет труда.

Часть проблем индексации является следствием различия представлений, поддерживаемых командой проекта. Например, в медицинском приборе требования восстановления после ошибки предохранительного устройства могут рассматриваться как “предохранительный” элемент одними членами команды и как элемент “исправления ошибки” другими членами. Обе точки зрения правильны; данные требования следует проиндексировать двумя способами.

Откровенно говоря, над индексацией приходится поработать, но, как правило, это нужно делать только один раз для каждого добавляемого в SRS нового требования. После этого элементы индекса существуют вместе с пакетом и становятся важной частью понимания проекта.

Индексирование следует использовать для того, чтобы иметь доступ к пакету, помимо TOC. Иными словами, не имеет смысла создавать индекс, элементы которого уже есть в оглавлении. Необходимо такое индексирование, чтобы читатель мог обратиться к понятиям, а не к заголовкам. Как и оглавления, индексы всегда следует обновлять в процессе публикации, чтобы обеспечить целостность пакета.

История исправлений

Крайне неприятно обнаружить, что вы просматривали устаревшую версию SRS. Каждая спецификация требований должна содержать страницу, посвященную истории исправлений, где хранится информация о соответствующих изменениях каждой версии элементов пакета. Как минимум, эта страница должна содержать следующую информацию.

- Номер исправления или код каждого изменения опубликованной информации
- Дату каждого исправления опубликованной информации
- Краткое описание произведенных исправлений
- Имя человека, ответственного за исправление

Кроме того, может оказаться полезным снабдить каждый изменяемый элемент SRS маркером исправлений. Отметки о внесении исправлений на полях помогают читателю найти изменения.

Большинство современных автоматических средств работы с документацией и требованиями обеспечивают мощные возможности контроля исправлений и автоматического ведения истории версий. Пользуйтесь ими!

Предостережение. Не устанавливайте контроль исправлений слишком рано (см. главу 34, "Управление изменениями"). Во время внесения исправлений одно и то же требование можно переделывать несколько раз, прежде чем его удастся удовлетворительно сформулировать. Не имеет смысла записывать все эти попытки; поэтому не включайте для пакета режим контроля исправлений, пока не достигнете относительной стабильности в процессе разработки программы. С другой стороны, не стоит слишком откладывать, в противном случае вас захлестнет неуправляемый процесс.

Глоссарий

Так как природа каждой прикладной области уникальна, в проектах со временем стремятся разработать специальный язык или систему сокращений. Чаще всего сокращения бывают мнемоническими, например "SRS". Их следует по возможности избегать. Также нужно воздерживаться от использования терминов, имеющих смысл только в контексте определенной ситуации. Хороший SRS содержит словарь таких терминов, чтобы помочь пользователям понять язык данной прикладной области. Следует позаботиться о том, чтобы включить в глоссарий объяснения всех специфических терминов проекта, аббревиатур и специальных фраз.