Глава 23

Требования к программному обеспечению

Основные положения

- Полный набор требований можно получить, определив вводы, выводы, функции и атрибуты системы, а также атрибуты ее окружения.
- В требования не следует включать общую информацию (графики, планы разработки, выделенные средства, тесты), а также информацию, касающуюся проектирования.
- Процесс разработки требований/проектирования является итеративным; выявленные требования ведут к выбору конкретных вариантов проектирования, которые в свою очередь приводят к возникновению новых требований.
- Ограничения проектирования касаются вариантов проектирования системы или процессов, с помощью которых система разрабатывается.

В предыдущих частях мы намеренно оставили определенные для системы функции на высоком уровне абстракции с тем, чтобы удобнее было выполнить следующие действия.

- Понять форму системы, сосредоточив внимание на ее функциях и на том, как они выполняют потребности пользователя.
- Оценить полноту и целостность системы, а также ее соответствие среде.
- Использовать данную информацию для определения возможности построения системы и управления ее масштабом перед тем, как будут производиться значительные инвестиции.

Кроме того, оставаясь на высоком уровне абстракции, мы воздерживаемся от преждевременного принятия чрезмерно ограничительных решений по требованиям, т. е. до того, как люди, непосредственно занимающиеся реализацией системы, получат возможность внести свой вклад в определение системы. В части 5, "Уточнение определения системы," мы переходим к разработке функций системы до уровня детализации, достаточного, чтобы гарантировать, что в ходе проектирования и кодирования будет создана система, полностью соответствующая потребностям пользователя. При этом мы переходим на следующий уровень конкретизации и создаем более полную, глубокую модель требований к разрабатываемой системе. Конечно, количество информации, которой надо управлять, также увеличивается, и нам необходимо подготовиться к работе с ней.

Уровень конкретизации, необходимый на этом шаге, зависит от множества факторов, среди которых содержание приложения и профессиональные навыки команды разработчиков. Для высокобезопасных систем программного обеспечения в медицине, авиации или интерактивной торговле уровень конкретизации особенно высок. Процесс уточнения может включать в себя использование формальных средств обеспечения качества, просмотры, ревизии, моделирование и т.п. В системах, где последствия сбоев не столь катастрофичны, уровень трудоемкости более умеренный. В этих случаях просто необходимо сформулировать определение системы достаточно четко, чтобы его могли понять все участники процесса, а также для того, чтобы обеспечить эффективные условия разработки и "достаточно высокую" вероятность успеха. Руководствуясь прагматическими и экономическими соображениями, следует создать достаточное количество спецификаций требований, чтобы разрабатываемая программа была именно тем, чего хочет пользователь.

Точно так же, как не существует языка программирования, подходящего для всех без исключения приложений, нет и универсального способа разработки более детальных спецификаций. В различных средах требуются различные методы, и тем, кто пишет требования и управляет ими, необходимо овладеть разнообразными профессиональными приемами. Нам приходилось в своей практике применять множество методов — от применения достаточно строгих документов требований, традиционных баз данных или архивов требований до использования моделей прецедентов и более формальных методов. Но всегда главное внимание уделялось спецификации на естественном языке, написанной достаточно ясно, чтобы ее могли понять все заинтересованные лица, заказчики, пользователи, разработчики и тестологи, но также достаточно конкретно ("Максимальная горизонтальная скорость оси 4 должна составлять 1м/с"), чтобы можно было выполнить верификацию и продемонстрировать соответствие. Перед тем как организовывать требования к системе, рассмотрим саму природу этих требований.

Определение требований к программному обеспечению

В главе 2 мы дали следующее определение требования.

- Некое свойство программного обеспечения, необходимое пользователю для решения проблемы при достижении поставленной цели.
- Некое свойство программного обеспечения, которым должна обладать система или ее компонент, чтобы удовлетворить требования контракта, стандарта, спецификации либо иной формальной документации.

Требования *к программному обеспечению* — это то, что данная программа делает для пользователя, прибора или другой системы. Начинать их поиск следует среди того, что "входит" в систему и "выходит" из нее, т.е. необходимо рассмотреть взаимодействия системы с ее пользователями.

Для этого проще всего сначала представить себе систему как некий черный ящик и подумать о том, что следует определить, чтобы полностью описать, что делает этот черный ящик.

Кроме входящей и выходящей информации, также необходимо обратить внимание на некоторые другие характеристики системы, в том числе на ее производительность и другие типы сложного поведения, а также на иные способы взаимодействия системы с ее средой (рис.23.1).

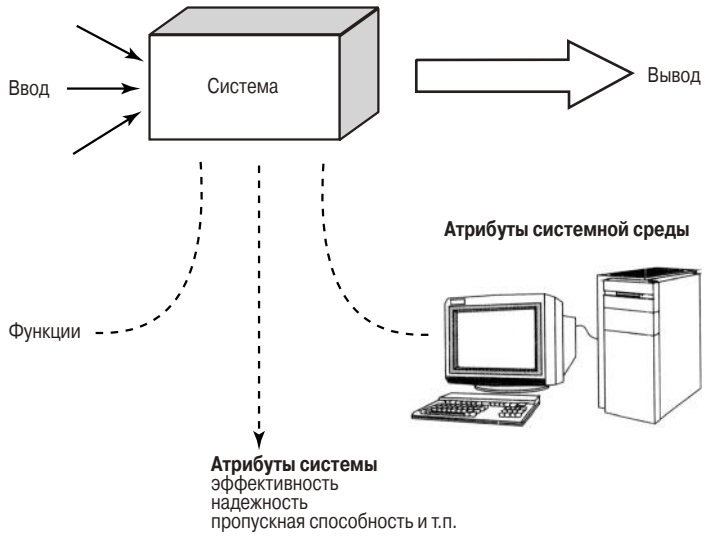


Рис. 23.1. Элементы системы

Используя аналогичный подход, Дэвис (Davis, 1999) отметил, что для полного определения системы необходимо описать следующие пять основных категорий элементов.

1. *Вводы системы.* Необходимо не только указать содержимое ввода, но и, если нужно, подробно описать устройства, а также протокол (форму, внешний вид и содержание) ввода. Как известно большинству разработчиков, этот класс может содержать значительный объем сведений и подвергаться частым изменениям, особенно в средах GUI, мультимедиа и Internet.
2. *Выводы системы.* Нужно описать поддерживаемые устройства вывода, такие как речевой вывод или видеотерминал, а также протокол и форматы генерируемой системой информации.
3. *Функции системы.* Отображение вводов в выводы и их различные комбинации.
4. *Атрибуты системы.* Типичные неповеденческие требования, такие как надежность, удобство сопровождения, доступность и пропускная способность, которые должны учитывать разработчики.
5. *Атрибуты системной среды.* Это такие дополнительные неповеденческие требования, как способность системы функционировать в условиях определенных операционных ограничений и нагрузок, а также совместимость с операционной системой.

На протяжении ряда лет мы использовали это разбиение на категории и убедились в его работоспособности. Оно способствует целостному и полному восприятию проблем требований. Таким образом, можно предложить следующее определение

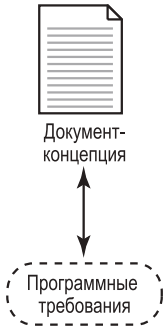
Полный набор требований к программному обеспечению можно задать, определив следующее:

- вводы системы;
- выводы системы;
- функции системы;
- атрибуты системы;
- атрибуты системной среды.

В результате мы сможем оценить, является ли некая "вещь" требованием к программному обеспечению, проверив, соответствует ли она данному подробному определению.

Взаимосвязь между функциями и требованиями к программному обеспечению

Ранее мы потратили некоторое время на изучение "функций" системы. Функции представляют собой описания желательного и полезного поведения. Теперь мы увидим, что существует соответствие между функциями и требованиями к программному обеспечению. В документе-концепции описаны функции на языке пользователя. Требования к программному обеспечению, в свою очередь, описывают эти функции более подробно. Чтобы предоставить пользователю некую функцию, разработчики должны выполнить одно или несколько конкретизированных программных требований. Другими словами, функции помогают понимать и обсуждать систему на высоком уровне абстракции, но с их помощью невозможно описать систему и создать на основании этого описания код. Для этой цели функции слишком абстрактны.



Требования к программному обеспечению более конкретизированы. Мы собираемся на их основе создавать код; следовательно, они должны быть "тестируемыми", т.е. достаточно конкретными, чтобы можно было проверить, действительно ли система реализует некое заданное требование. Предположим, мы разрабатываем систему обнаружения неполадок для конвейерного производства или организации, разрабатывающей программное обеспечение. В табл. 23.1 представлена некая функция документа-концепции и связанный с ней набор требований. Эта связь и возможность осуществлять трассировку функций к требованиям (и наоборот) лежат в основе очень важного понятия в области управления требованиями, известного как "трассируемость" (traceability). (Эту тему мы будем обсуждать позднее.)

Таблица 23.1. Требования, связанные с некоторой функцией документа-концепции

Документ-концепция	Программные требования
Функция 63. Система обнаружения неполадок будет предоставлять информацию об обнаруженных дефектах, чтобы помочь пользователю оценить состояние проекта	SR63.1. Информация будет предоставляться в виде отчета-гистограммы, где по оси *x* откладывается время, а по оси *y* — количество обнаруженных дефектов

Окончание табл. 23.1

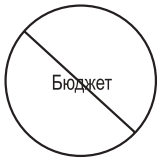
Документ-концепция	Программные требования
	SR63.2. Пользователь может задавать временной период в днях, неделях или месяцах
	SR63.3. Пример отчета об обнаруженных дефектах представлен на прилагаемом рисунке

Дилемма требований: что или как

Требования сообщают разработчикам, что должна делать система, и должны описывать системные вводы, выводы, функции, атрибуты, а также атрибуты системной среды. Но существует много другой информации, которую требования *не должны* содержать. В частности, не следует указывать не являющиеся необходимыми подробности проектирования или реализации, а также информацию, связанную с управлением проектом. Кроме того, не следует описывать, как система будет тестироваться. Тогда требования будут сосредоточены на поведении системы и будут изменяться только тогда, когда меняется поведение.

Исключение информации, связанной с управлением проектом

Иногда из соображений удобства менеджер проекта может включить в набор требований информацию, связанную с управлением проектом, а именно графики, планы управления конфигурацией, планы испытаний, бюджеты и штатные расписания. Как правило, этого следует избегать, так как изменения в этой информации (например, изменения графика) будут приводить к необходимости замены устаревших "требований". (Когда требования датируются, им меньше доверяют и чаще игнорируют.) Кроме того, неизбежные дебаты относительно вопросов управления проектом следует отделить от обсуждения того, *что должна делать система*. Существуют различные заинтересованные лица, и у каждого из них свои цели.



Бюджет тоже может выглядеть как требование; тем не менее это информация другого рода, не удовлетворяющая нашему определению и, следовательно, не относящаяся к системным или программным требованиям. Бюджет — важная информация, особенно когда разработчики решают, какую стратегию реализации избрать, поскольку некоторые стратегии могут быть слишком дорогостоящими, а время их выполнения слишком длительным. И все же это не требование. Аналогично описание процедур тестирования или приемки (с помощью которых мы будем определять, что требования действительно выполнены) также не удовлетворяет определению и, следовательно, не является требованием.

Как следует из нашей практики, все же иногда полезно включить некую дополнительную информацию. Например, часто составитель требования может "намекнуть", какой тест подойдет для разработанного им требования. В конечном счете, он имеет представление о том, какое именно поведение соответствует данному требованию, и разумно воспользоваться его помощью.

Исключение информации, относящейся к проектированию

Требования также не должны содержать информацию о системной архитектуре и техническом проекте. В противном случае можно ненамеренно ограничить команду в выборе наиболее подходящих для данного приложения вариантов проектирования. ("Эй, мы должны проектировать его таким способом; так сказано в требованиях.")

Исключить из требований детали, относящиеся к управлению проектом и тестированию, достаточно просто. Но исключение деталей проектирования и реализации обычно гораздо более сложная и тонкая работа. Предположим, что первое требование в табл. 23.1 сформулировано следующим образом: "SR63.1. Информация об обнаруженных дефектах будет предоставляться в виде отчета-гистограммы, написанного на Visual Basic, причем основные причины будут откладываться по оси *x*, а количество дефектов – по оси *y*" (рис. 23.2).

Рис. 23.2. Отчет-гистограмма

Хотя указание Visual Basic в качестве языка написания программы является достаточно явным нарушением наших рекомендаций (поскольку оно не представляет ни ввод, ни вывод, ни функцию, ни атрибут поведения), полезно задать вопрос: "Кто принял решение, что гистограмма должна быть реализована на Visual Basic, и почему оно было принято?" Возможны следующие ответы.

- Один из технически ориентированных членов группы, определяя документ-концепцию, решил, что следует указать Visual Basic, так как это "наилучшее" решение проблемы.
- Это может быть указание пользователя. Опасаясь, что разработчики могут применить некую более дорогостоящую и менее доступную технологию, пользователь решает, что технология VB – доступная и относительно дешевая, и хочет, чтобы использовали именно ее.
- Политическое решение, принятое организацией-разработчиком, может диктовать, чтобы все приложения разрабатывались на Visual Basic. Чтобы пресечь все попытки проигнорировать данную политику, руководство настаивает, чтобы упоминание Visual Basic было повсюду, где это возможно, включено в документы требований.

Если технический разработчик решил включить упоминание о Visual Basic, поскольку просто предпочитает данный язык, оно, бесспорно, не является легитимным элементом списка требований. Если это требование предложено пользователем, ситуация несколько

иная. Если клиент отказывается платить за систему, написанную не на Visual Basic, лучше всего трактовать подобное пожелание как требование, *хотя мы поместим его в специальный класс, называемый ограничениями проектирования, так что оно будет отделено от обычных требований, описывающих только внешнее поведение.* Тем не менее это именно ограничение реализации, налагаемое на команду разработчиков. (Кстати, если вы думаете, что этот пример нереалистичен, напомним, что до конца 1990-х обычным требованием Министерства обороны США к разработчикам программного обеспечения было "создавать системы, используя язык Ada".)

Однако рассуждая в этом примере о Visual Basic, мы можем пропустить более тонкий и, возможно, более важный момент анализа требования: *почему информация об обнаруженных дефектах должна предоставляться в виде отчета-диаграммы?* Почему не круговая диаграмма или другое представление информации? Подразумевает ли слово "отчет" некий бумажный документ или информация может отображаться на экране компьютера? Нужно ли хранить эту информацию с тем, чтобы ее можно было передавать другим программам или пересылать в корпоративную сеть? Функцию, описанную в документе-концепции, практически всегда можно осуществить множеством способов, причем некоторые из них имеют совершенно определенные реализационные последствия.

Во многих случаях на описание проблемы, которое служит основой формулирования требования, влияют представления пользователя о возможных способах ее решения. То же верно и по отношению к разработчикам, которые совместно с пользователем участвуют в процессе формулирования функций документа-концепции и требований. Как гласит старая пословица, если ваш единственный инструмент — молоток, все ваши проблемы похожи на гвозди. Но нам нужно проявлять бдительность, чтобы необязательные ограничения реализации не попадали в требования, и мы должны удалять подобные ограничения везде, где это возможно.

Больше внимания требованиям, а не проектированию

До сих пор мы трактовали требования к программному обеспечению, решения и ограничения проектирования так, как если бы они были различными сущностями, которые можно четко разграничить. Иными словами, мы предполагали следующее.

- Разработка требований предшествует проектированию.
- Пользователи и заказчики принимают решения относительно требований.
- Разработчики принимают решения относительно проектирования, так как они лучше подготовлены к тому, чтобы выбрать среди многих вариантов проектирования наиболее подходящий для удовлетворения конкретной потребности.

Это удачная модель, от которой можно отталкиваться при решении задач управления требованиями. Дэвис (Davis, 1993) назвал ее парадигмой "что вместо как", где "что" — это требования ("что" система должна делать), а "как" представляет собой проектное решение, которое следует реализовать для достижения этой цели.

Для такого представления есть причины. Действительно, лучше понять требования перед тем, как приступать к проектированию, а боль-

шинство ограничений проектирования ("использовать XYZ-библиотеку классов для доступа к базе данных") являются важными проектными решениями, записанными в активы требований с тем, чтобы мы знали, что они получены по условиям контракта или по достаточно веским техническим причинам.

Если мы не сможем произвести такую классификацию вовсе, картина будет очень запутанной, и мы не сумеем отделить разработку требований от проектирования. Более того, мы не будем знать, кто за что отвечает в процессе разработки. Или того хуже, наши заказчики будут диктовать нам проектные решения, а разработчики — требования.

Однако существует пока невидимая, но достаточно серьезная проблема, которая противоречит представленной нами простой парадигме. Вернемся к нашему рабочему примеру. Если команда принимает проектное решение использовать ПК-технологию для подсистемы "Центральный блок управления" (ЦБУ) системы HOLIS, это, вероятно, окажет некое внешнее воздействие на пользователя (он будет видеть подсказку или экран-приглашение). Если мы захотим воспользоваться некоторыми возможностями ОС, то соответствующие библиотеки классов будут, конечно же, демонстрировать внешнее поведение пользователю. (Заметим, что его можно скрыть, но это не нужно.)

Если воспользоваться предложенным в данной главе определением, возникает вопрос: *если проектное решение воздействует на внешнее поведение, заметное пользователю, становится ли такое решение (явно воздействующее на ввод или вывод системы) требованием?* Ответ ("да", "нет" или "не имеет значения") будет зависеть от вашей интерпретации определения и проведенного нами анализа. Но это проливает свет на очень важный аспект, так как его понимание жизненно необходимо для уяснения природы итеративного процесса в целом. Давайте рассмотрим его более подробно.

Итерационный цикл разработки требований и проектирования

В реальности деятельности по разработке требований и проектированию должны осуществляться итеративно. *Выявление требований, их определение и принятие проектных решений циклически чередуются.* Процесс заключается в непрерывном круговороте.

> Имеющиеся требования приводят к выбору определенных вариантов проектирования,
> а
> те в свою очередь могут инициировать новые требования.

Иногда появление новой технологии может привести к тому, что мы отбросим множество предположений о том, какими должны быть требования; мы можем найти совершенно иной подход, перечеркивающий старую стратегию. ("Давайте целиком уберем модуль *клиент/доступ к данным/GUI* и заменим его навигационным интерфейсом.") Это важный и правомерный источник изменения требований.

Такой процесс закономерен, попытка поступать по-другому будет безрассудством. С другой стороны, во всем этом есть серьезная опасность: если мы не достигли истинного понимания потребностей заказчика *и* не привлекали его к процессу разработки требований (а иногда даже и к процессу понимания наших действий по *проектированию*), может быть принято *неверное* решение. При правильном осуществлении процесс "непрерывного пересмотра требований и проектирования" является просто фантастическим, так как позволяет постоянно совершенст-

вовать нашу способность удовлетворить реальные потребности клиентов. *Именно в этом и состоит суть эффективного итеративного управления требованиями.* Но если данный процесс осуществляется неправильно, мы постоянно "гоняемся за хвостом нашей технологии", и результаты весьма плачевны. Мы никогда не говорили, что это будет легко.

Дальнейшая характеристика требований

Итак, существуют различные "разновидности" требований. В частности, мы считаем полезным выделить следующие три типа (рис. 23.3).

- Функциональные требования к программному обеспечению
- Нефункциональные требования к программному обеспечению
- Ограничения проектирования

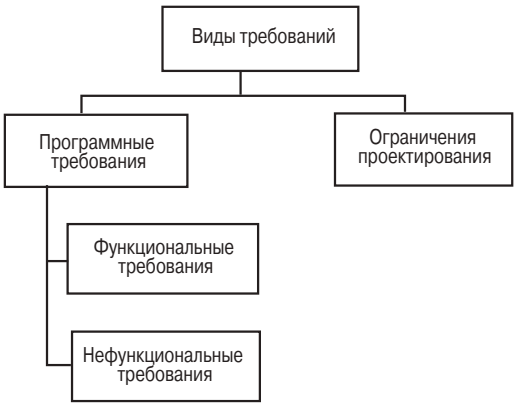


Рис. 23.3. Типы требований

Функциональные требования к программному обеспечению

Функциональные требования описывают, как ведет себя система. Эти требования обычно ориентированы на действия ("Когда пользователь делает *x*, система будет делать *y*"). Большинство продуктов и приложений, предназначенных для выполнения полезной работы, содержит множество функциональных требований к программному обеспечению. Программное обеспечение используется для реализации большей части функциональных возможностей.

При определении функциональных требований следует искать золотую середину между слишком конкретизированной формулировкой требования и слишком общей и неоднозначной. Например, как правило, ни к чему иметь обобщенное функциональное требование следующего вида: "Когда вы нажимаете эту кнопку, система включается и работает". С другой стороны, формулировка требования, которая состоит из нескольких страниц текста, по-видимому, слишком конкретизирована, но может быть правомерной в некоторых частных случаях. Мы вернемся к рассмотрению этого вопроса в главе 26.

Оказывается, что большинство функциональных требований можно сформулировать в виде простых декларативных определений или в форме прецедентов, которым посвя-

щена следующая глава. Опыт свидетельствует, что определение требования, состоящее из одного-двух предложений, обычно является наилучшим, когда нужно соотнести потребность пользователя с приемлемым для работы разработчика уровнем конкретизации.

Нефункциональные программные требования

До сих пор в данной главе приводились, в основном, примеры *поведенческих (функциональных)* требований к системе, основное внимание в которых уделялось вопросам ввода, вывода и обработки. Функциональные требования описывают, как система должна вести себя, когда ей предоставляются определенные входные данные или условия.

Но этого недостаточно для полного описания требований к системе. Необходимо также учитывать следующие характеристики, которые Грейди (Grady) (1992) назвал *"нефункциональными требованиями"*.

- Практичность(Usability)
- Надежность(Reliability)
- Производительность(Performance)
- Возможность обслуживания(Supportability)

Эти требования, как правило, используются для описания некоторых "атрибутов системы" или "атрибутов системного окружения" из нашего сложного определения. Благодаря их удобной классификации мы можем больше узнать о системе, которую необходимо создать. Рассмотрим каждый из перечисленных пунктов более подробно.

Практичность

Важно описать, насколько легко будущие пользователи смогут освоить данную систему. Может понадобиться определить различные категории пользователей: начинающие, "средние", опытные, а также неграмотные пользователи — те, которые не владеют свободно родным языком "средних" пользователей, и т.д. Если предполагается, что заказчик будет просматривать требования и участвовать в их обсуждении (а так и должно быть!), следует все требования в этой сфере писать на естественном языке (описание практичности не может быть в форме конечного автомата).

Поскольку практичность зависит от точки зрения наблюдателя, как описать такой неясный набор требований? Ниже приводятся некоторые рекомендации.

- Указать необходимое время подготовки пользователя для достижения минимальной производительности (способности выполнять простые задачи) и операционной производительности (способности выполнять обычные текущие задачи). Как отмечалось, могут понадобиться отдельные описания для начинающих пользователей (которые, быть может, никогда прежде не видели компьютер), средних и опытных.
- Указать время выполнения типичных задач или транзакций, осуществляемых конечным пользователем. При создании системы ввода заказов на покупку, вероятно, наиболее типичными задачами, выполняемыми конечными пользователями, будут ввод, удаление или модификация заказа, а также проверка состояния заказа. Если пользователь знает, как выполнять эти задачи, сколько времени он потратит на ввод типичного заказа? Минуту? Пять минут? Час? Конечно, это будет зависеть от технической реализации (скорости модема, пропускной способности сети, ОП, мощности ЦП, которые совместно определяют время ответа, обеспечиваемое сис-

темой), но время выполнения задач также напрямую зависит от практичности системы, и нужно иметь возможность определить это отдельно.

- Сравнить практичность новой системы с уже существующими современными системами, которые известны и пользуются успехом у пользователей. Иными словами, требование может выглядеть так: "Новая система должна быть признана подавляющим большинством (90%) пользователей такой же практичной, как и существующая XYZ-система". Напоминаем, что такие требования, равно как и все остальные, должны допускать возможность проверки соответствия; мы должны описывать требование так, чтобы пользователи могли проверить соответствие практичности установленному нами критерию.
- Оговорить существование систем интерактивных подсказок, программ-помощников, средств предупреждения, руководств пользователя и других форм документации и помощи, а также определить их необходимые функции.
- Следовать соглашениям и стандартам, разработанным для человеко-машинного интерфейса. Чтобы новая система работала "почти так же, как та, которую я уже использовал", необходимо следовать согласованным стандартам от приложения к приложению. Например, вы можете указать требование соответствия общим стандартам практичности, таким как стандарты CUA (Common user access) компании IBM или стандарты Windows-приложений, опубликованные компанией Microsoft.

Примером подлинного прорыва в сфере практичности в компьютерном мире может служить разница между командными строками операционных систем DOS и UNIX и GUI-интерфейсами операционных систем Windows и Macintosh. GUI-интерфейсы значительно упростили использование компьютера для пользователей, не имеющих технического образования. Еще одним примером является Internet-броузер, который стал окном в мир World Wide Web и радикально ускорил освоение Internet средним пользователем.

Было предпринято несколько интересных попыток сделать более строгим весьма расплывчатое понятие практичности. Наибольший интерес представляет так называемый "Билль о правах пользователей" (Карат (Karat), 1998). Последняя его версия состоит из десяти основных пунктов.

1. Пользователь всегда прав. Если возникает проблема с использованием системы, то дело в системе, а не в пользователе.
2. Пользователь имеет право на программное и аппаратное обеспечение, которое легко монтируется и демонтируется без негативных последствий.
3. Пользователь имеет право на то, чтобы система делала в точности то, что обещано.
4. Пользователь имеет право на простые в использовании инструкции (руководства пользователя, интерактивные или контекстно-зависимые подсказки, сообщения об ошибках), которые позволят ему понимать систему и использовать ее для достижения желаемых целей, а также эффективно и легко выходить из сложных ситуаций.
5. Пользователь имеет право на внимание со стороны системы, а также на то, чтобы иметь возможность получить ответ системы на запрос о внимании.
6. Пользователь имеет право на то, чтобы система предоставляла четкую, понятную и точную информацию о выполняемой задаче и ее выполнении.

7. Пользователь имеет право на то, чтобы его информировали о всех системных требованиях для успешного использования программного обеспечения или аппаратуры.
8. Пользователь имеет право знать о пределах возможностей системы.
9. Пользователь имеет право общаться с провайдером технологии и получать полные исчерпывающие ответы, когда в этом возникает необходимость.
10. Пользователь должен быть хозяином программных и аппаратных технологий, а не наоборот. Продукты должны использоваться естественно и интуитивно.

Отметим, что некоторые из перечисленных пунктов по своей сути являются неизмеримыми и не могут быть кандидатами в требования. С другой стороны, ясно, что этот документ может служить отправной точкой при разработке вопросов и определении требований, касающихся практичности предлагаемого продукта.

Надежность

Конечно, никому не нравятся ошибки, системные сбои или потери данных, и если подобные явления нигде в требованиях не упоминаются, пользователь, естественно, может предположить, что их не будет вовсе. Но в современном мире даже самый оптимистически настроенный пользователь знает, что ошибки и сбои неизбежны. Таким образом, в требованиях следует указать, в какой степени система *обязана* вести себя приемлемым для пользователя образом. Как правило, здесь описываются следующие аспекты.

- *Доступность (availability).* Система должна быть доступна для операционного использования в течение указанного времени (в процентном выражении). Иногда требование может указывать непрерывную "nonstop" доступность, т.е. 24 часа в сутки, 365 дней в году. Но чаще можно встретить указание 99-процентной или 99.9-процентной доступности между 8 часами утра и полуночью. Отметим, что требования должны указывать, что понимается под "доступностью". Означает ли 100-процентная доступность, что все пользователи должны иметь возможность использовать все системные услуги в любое время?
- *Среднее время между отказами (Mean time between failures, MTBF).* Оно обычно указывается в часах, но может также указываться в днях, месяцах или годах. Здесь тоже нужна точность: требования должны четко определять, что понимается под "сбоем".
- *Среднее время восстановления (mean time to repair, MTTR).* Как долго системе разрешается не работать после сбоя? MTTR может иметь несколько значений; например, пользователь может указать, что 90% всех системных сбоев должны ликвидироваться за 5 минут, а 99.9% — в течение часа. Вновь важна точность: требования должны четко указывать, означает ли восстановление, что все пользователи снова будут иметь возможность получать доступ ко всем услугам, или допускается частичное восстановление.
- *Точность (accuracy).* Какая точность требуется системам с числовым выводом? Например, должны ли результаты в финансовых системах быть с точностью до пенни или доллара?
- *Максимальный коэффициент ошибок.* Как правило, выражается как число ошибок, приходящееся на тысячу строк кода (ugs/KLOS) или на одну функцию.

- *Количество различных ошибок.* Обычно ошибки делятся на незначительные, серьезные и критические. Здесь также важны четкие определения: требования должны определять, что понимается под "критической" ошибкой — полная потеря данных или невозможность использовать определенную часть функциональных возможностей системы.

В некоторых случаях требования могут указывать некоторые "ориентировочные" метрики надежности. Типичным примером этого является использование некой "метрики сложности" для оценки сложности и, тем самым, потенциальной "ошибочности" программы.

Производительность

К требованиям производительности обычно относится следующее.

- Время ответа для транзакции: среднее, максимальное
- Пропускная способность: число транзакций в секунду
- Емкость: сколько пользователей или транзакций может обслужить система
- Режимы снижения производительности: допустимые режимы работы при ухудшении параметров системы

Если новая система должна совместно с другими системами или приложениями использовать аппаратные ресурсы (ЦП, память, каналы, дисковую память, сетевой диапазон частот), может также потребоваться указать, насколько "цивилизованно" она себя при этом ведет.

Возможность обслуживания

Возможность обслуживания заключается в способности легко модифицировать программное обеспечение с целью внесения изменений и исправлений. В некоторых предметных областях можно заранее предвидеть вероятную природу будущих изменений, и требования могут содержать указание "времени ответа" группы поддержки для простых, средних и сложных изменений.

Предположим, мы создаем новую систему расчета заработной платы. Одно из многих требований к такой системе состоит в том, что она должна вычислять удерживаемые правительственные налоги для каждого работника. Пользователь, конечно, знает, что правительство каждый год меняет алгоритм вычисления налогов. Это изменение затрагивает две величины: вместо удержания X процентов от общей заработной платы работника, но не более $P, новый закон требует удержания Y процентов, но не более $Q. Таким образом, требование можно сформулировать так: "Модификации системы с целью задания новых коэффициентов налогообложения должны осуществляться командой в течение одного дня после получения уведомления от официальных властей".

Но предположим, что налоговая инспекция периодически вносит в данный алгоритм поправки, аналогичные следующей: "Для левшей с голубыми глазами ставка налогообложения должна составлять Z процентов, но не более $R". Подобные модификации сложнее предусмотреть в программе; хотя можно попытаться сделать ее максимально гибкой. Команда, вероятно, согласится, что данная модификация попадает в категорию изменений "среднего уровня" сложности, для которых требование может задавать время реагирования — одна неделя.

Представим себе, что перед началом проекта менеджер отдела заработной платы сказал: "Возможно, мы будем расширять сферу нашей деятельности. В этом случае нужно

иметь возможность сделать так, чтобы алгоритм вычисления удерживаемого налога отражал действующее законодательство Франции, Германии или Гонконга". Если предположить, что такое "требование" вообще имеет смысл, его можно сформулировать только в виде намерений и целей; и будет сложно измерить и проверить его выполнение. Чтобы действительно повысить вероятность возможности обслуживания системы в данной ситуации, нужно потребовать использования определенных языков программирования, систем управления базами данных (СУБД), программных средств, стандартных процедур поддержки, стилей и стандартов программирования и т.д. (В этом случае, как мы увидим далее, требования, в действительности, становятся ограничениями проектирования.) Нельзя утверждать, что в результате система станет легко обслуживаемой, но, по крайней мере, мы можем приблизиться к цели.

Ограничения проектирования

Ограничения проектирования, как правило, касаются вариантов проектирования системы или процессов, используемых при ее построении. Ниже кратко описаны различные формы ограничений проектирования.

- Некое требование, которое допускает несколько вариантов проектирования; проект является осознанным выбором среди этих вариантов. Если это возможно, хотелось бы не указывать конкретный вариант в требованиях, а оставить выбор за разработчиками, так как они смогут лучше оценить технические и экономические характеристики каждого варианта. Если мы не оставляем возможности выбора ("Использовать СУБД Oracle"), возможности проектирования сужаются, утрачивается гибкость и свобода разработки.
- Требование, налагаемое на процесс создания программы ("Программировать на VB" или "ИспользоватьXYZ-библиотеку классов").

Как было показано в примере с Visual Basic, источники и причины таких ограничений могут быть различны, и разработчики иногда вынуждены принимать их, независимо от того, нравятся они им или нет. Но важно отделять их от обычных требований, так как подобные ограничения могут быть достаточно произвольными, они могут быть обусловлены политическими соображениями, а также могут подвергаться изменениям по мере развития технологий.

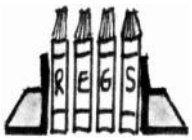
Рассмотрим определение.

> *Ограничения проектирования налагаются на проект системы или процессы, с помощью которых система создается. Они не влияют на внешнее поведение системы, но должны выполняться для удовлетворения технических, деловых или контрактных обязательств.*

Можно указать следующие источники ограничений проектирования

- Операционные среды:*Программы пишутся на Visual Basic.*
- Совместимость с существующими системами: *Приложение должно выполняться как на новой, так и на прежней нашей платформе.*
- Прикладные стандарты: *Использовать библиотеку классов из Developer's Library 99-724 на корпоративном сервере IT.*

- Корпоративные практические наработки и стандарты: *Должна обеспечиваться совместимость с существующей базой данных, Использовать стандарты программирования C++.*



Еще одним важным источником ограничений проектирования являются разнообразные инструкции и стандарты, которым подчиняется разработка проекта. Например, разработка медицинских продуктов в США регулируется множеством инструкций и стандартов FDA (Управление по санитарному надзору за продуктами и медикаментами), которые касаются не только продукта, но и процесса его разработки и документирования. Среди организаций, инструкциям и стандартам которых должно отвечать проектирование, можно указать следующие.

- Управление по санитарному надзору за продуктами и медикаментами (FDA)
- Федеральная правительственная комиссия США по средствам связи (FCC)
- Министерство обороны (DOD)
- Международная организация по стандартизации (ISO)
- Лаборатории по технике безопасности (UL)

Как правило, сформулированные в виде разнообразных инструкций проектные ограничения этого типа слишком длинные, чтобы их можно было непосредственно включить в требования. В большинстве случаев достаточно внести их в список ограничений проектирования в форме *ссылок*. Таким образом, требование может иметь вид: *Программа будет полностью соответствовать стандарту TüV Software Standard, разделы 3.1-3.4.*

Однако при включении подобных ссылок тоже имеется определенный риск. Нужно внимательно следить за тем, чтобы включать конкретные, имеющие отношение к делу, а не общие ссылки. Например, одна ссылка вида *Продукт должен соответствовать стандарту ISO 601* фактически "свяжет" ваш продукт со всеми стандартами документа. Обычно следует стремиться найти "золотую середину" между чрезмерной и недостаточной конкретизацией.

Практически все проекты будут иметь те или иные ограничения проектирования. При работе с ними мы предлагаем руководствоваться следующими рекомендациями.

- Следует отличать их от других требований. Например, если программные требования обозначены ярлыком "SR" (software requirement), для ограничений проектирования можно использовать ярлык "DC" (design constraint). Можно также попытаться различать истинные ограничения проектирования и регулирующие ограничения, но мы пришли к выводу, что это редко бывает полезно и может привести к непомерным затратам на поддержку.
- Лучше включить все ограничения проектирования в специальный раздел пакета требований или использовать специальный атрибут, чтобы их можно было легко собрать вместе. Это позволит при необходимости легко находить их и пересматривать.
- Необходимо указывать источник каждого ограничения проектирования. Тогда вы сможете позже вернуться к обсуждению этого требования. "Так, это ограничение поступило от Билла из отдела маркетинга. Давайте поговорим с ним об этом!" Если имеются ссылки на некие стандарты, следует составить специальную библиографическую справку. Тогда в будущем будет проще найти данный стандарт.

- Следует документировать объяснения для каждого ограничения проектирования. Записывайте одно-два предложения, объясняющие, почему то или иное ограничение проектирования налагается на проект. Это поможет вам позднее вспомнить, что послужило мотивом для наложения конкретного ограничения. Как следует из нашего опыта, практически всегда возникает вопрос: "Почему мы наложили здесь это ограничение?". Документирование пояснений позволяет более эффективно работать с ограничениями проектирования на более поздних фазах проекта, когда о них (неизбежно) зайдет речь.

Являются ли ограничения проектирования истинными требованиями?

Можно считать, что ограничения проектирования не являются требованиями к программному обеспечению, так как они не представляют ни один из пяти пунктов нашего сложного определения. Но когда ограничение проектирования поднимается до уровня полноправного политического, технического или бизнес-условия, оно будет удовлетворять нашему определению, как нечто, необходимое для "соответствия контракту, стандарту, спецификации или другой формальной документации".

В таких случаях проще всего трактовать ограничение проектирования так, как любое другое требование, и удостовериться, что система проектируется и разрабатывается в соответствии с этим ограничением. Однако всегда следует стремиться к тому, чтобы таких ограничений было как можно меньше, так как их наличие может зачастую ограничить наш выбор при реализации других требований, непосредственно выполняющих потребности пользователя.

Поучительная история

Мы работали с компанией Fortune 500, хорошо известной в отрасли благодаря ее приверженности процессу и процедуре. Представьте себе наше удивление, когда мы обнаружили, что работа компании по сбору требований полностью парализована из-за того, что команда не может прийти к согласию о том, являются ли определенные требования функциональными, нефункциональными или ограничениями проектирования. В результате способность команды двигаться вперед в осуществлении проекта оказалась под вопросом из-за игры слов! Мы сказали команде, что неважно, как это называется, давайте продвигаться хоть в чем-нибудь. Мораль такова. Назначение классификации в том, чтобы стимулировать мышление, помогать при поиске "неоткрытых руин", и в том, чтобы помочь по-разному воспринимать эти вещи. Но в действительности классификация не имеет значения, если вы понимаете, что требования — это нечто, с чем вас или систему будут сравнивать. Предпочтительнее двигаться вперед (пусть и с несовершенной организацией), чем топтаться на месте, разрабатывая план совершенного разбиения требований по категориям.

Использование “дочерних” требований для повышения уровня конкретизации

Мы обнаружили, что многие проекты выиграют от использования концепции *дочерних требований* в качестве средства дополнения неких базовых требований. *Дочернее требование служит для повышения уровня конкретизации, выраженного в родительском требовании.*

Рассмотрим пример. В этот раз мы будем использовать в качестве иллюстрации пример из области аппаратного обеспечения. Предположим, вы разрабатываете электронный прибор, работающий от стандартной электросети. Иными словами, пользователь собирается поместить прибор в отверстие в стене. Возникает вопрос: “Как сформулировать требования, касающиеся питания данного прибора?

Совершенно естественным является следующее требование: “Прибор должен работать от стандартной электросети Северной Америки”. Но что это значит? Ваши инженеры забросают вас вопросами о напряжении, токе, частоте и т.д. Конечно, вы можете переписать требование так, чтобы оно содержало все необходимые детали, но вы, вероятно, обнаружите, что включение всех технических характеристик скрыло первоначальную цель требования. В конце концов, вы просто хотели, чтобы прибор работал, если его поместить в отверстие в стене!

В таком случае вы можете создать несколько требований для задания напряжения, тока, частоты и т.д. Эти требования следует рассматривать как “дочерние” определенного родительского требования. В дальнейшем мы будем часто использовать отношения “родитель-ребенок” в иерархической структуре требований. Таким образом, спецификация энергетических потребностей данного прибора будет выглядеть следующим образом.

Родительское требование. Прибор должен работать от стандартной электросети Северной Америки.

Дочернее 1. Прибор должен работать при напряжении в диапазоне xxx–yyy вольт AC.

Дочернее 2. Для нормального функционирования прибор должен потреблять не более xxx AC ампер.

Дочернее 3. Прибор должен работать так, как указано в спецификации, если входная частота варьируется в пределах xx–yy герц.

Использование дочерних требований является способом гибкого расширения и дополнения спецификации и одновременно обеспечивает контроль глубины представляемых деталей. В нашем примере естественно представить спецификацию верхнего уровня в таком виде, чтобы пользователи могли легко понять ее. В то же время разработчики могут просмотреть подробные дочерние спецификации, чтобы убедиться, что они понимают все детали реализации.

Это понятие можно использовать и в том случае, если необходима дальнейшая конкретизация. Например, легко представить себе ситуацию, когда “дочернее” требование в свою очередь становится “родительским” для следующего уровня детализации. Другими словами, можно расширять иерархию далее, до такого уровня детализации, в котором нуждается продукт.

Родительское:

Дочернее1:

Внучатое 1:
Внучатое 2:

Но и здесь необходимо сделать некое предостережение. Хотя понятие дочерних требований чрезвычайно полезно, необходимо избегать слишком большого числа иерархических уровней детализации, просто потому, что вы запутаетесь в массе микроскопических деталей и потеряете перспективу основной цели пользователя. Как следует из нашего опыта, для большинства проектов достаточно одного подуровня детализации. В крайнем случае может оказаться полезным перейти к двум подуровням — "дочернему" и "внучатому" — но вряд ли понадобится спускаться ниже этого уровня детализации.

Организация дочерних требований

Мы обнаружили, что лучше всего не отделять дочерние требования от родительских, а включать их в главный пакет требований.

Чтобы при чтении проще было связать дочерние требования с родительским, обозначение дочерних требований должно основываться на обозначении родительских. Предположим, что требование к программному обеспечению SR63.1 (см. табл. 23.1) имеет одно или несколько дочерних требований. Естественно будет обозначить дочерние требования SR63.1.1, SR63.1.2, SR63.1.3 и т.д. Иерархический вид табл. 23.1 будет тогда выглядеть следующим образом.

Функция 63
  SR63.1
    SR63.1.1
    SR63.1.2
    SR63.1.3
  SR63.2

При работе в среде программных/дочерних требований полезно иметь возможность расширять/сужать полный набор требований так, чтобы можно было рассматривать только родительские требования (отдельно) или родительские вместе с дочерними.

Далее...

После того как мы изучили природу требований, можно переходить к методам их *фиксации* и *организации*. Следующая глава будет посвящена мощному методу фиксации требований. Последующие главы посвящаются вопросам организации коллекции требований.