Элизабет Халл
Кен Джексон
Джереми Дик

Разработка и управление требованиями

Практическое руководство пользователя

(Второе издание)

2005

Об авторах и книге

Elizabeth Hull, BSc, PhD, CEng, FBCS
School of Computing and Mathematics, University of Ulster
Newtownabbey, Co Antrim, UK

Kenneth Jackson, BSc, MSc, MBCS
Telelogic UK Ltd., Oxford, UK

Jeremy Dick, BSc(Eng), ACGI, DPhil, DIC, MA
Telelogic UK Ltd., Oxford, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

ISBN 1-85233-879-2 Springer London Berlin Heidelberg
Springer Science+Business Media
springeronline.com

Typesetting: Gray Publishing, Tunbridge Wells, Kent
Printed and bound in the United States of America
34/3830-543210 Printed on acid-free paper SPIN 10981719

От переводчика

Уважаемые коллеги,

*Да..., именно так – **коллеги**. И это не просто формальное обращение...*

Я не являюсь профессиональным переводчиком, а занимаюсь анализом и разработкой требований. Вот уже почти 10 лет моя профессиональная деятельность тесно связана с телекоммуникациями и программным обеспечением. Стремление выполнять свою работу лучше заставляет меня постоянно учиться - читать новые публикации, участвовать в конференциях, обмениваться опытом и мнениями с коллегами.

Прочитав эту книгу в оригинале, я просто не мог ее не перевести. Она буквально заставила меня отложить свои дела и посвятить свободное время переводу. Книга привлекла меня, в первую очередь, своим фундаментальным академическим подходом, что в англоязычной литературе в компьютерной области встречается не так часто.

На мой взгляд, авторам удалось невероятно элегантно скомпоновать материал книги. В первой и второй главах излагаются основные идеи, которые развиваются на протяжении последующих глав. Нигде ранее я не встречал попыток выделить общие методы и подходы, применимые для разных этапов разработки систем. Введение в общий процесс разработки является краеугольным камнем книги, а метод расширенных связей выводит разработку требований на совершенно иной качественный уровень.

Работая над переводом книги, я надеялся сделать ее доступной более широкому кругу свои коллег, а так же тем молодым и талантливым людям, которые только вступают на этот путь.

Огромное спасибо авторам книги, компании Telelogic и лично сотруднику ее российского офиса Анатолию Волохову.

Буду рад получить ваши замечания и комментарии, обменяться мнениями и идеями.

С уважением,
Илья Корнипаев
ilya_kornipaev@mail.ru

Предисловие ко второму изданию

Выпуск второго издания книги, спустя совсем небольшое время после выхода в свет первого издания, является свидетельством того, что тема, которой посвящена книга, развивается очень стремительно. Второе издание включает те значительные изменения, которые произошли за два года после выхода в свет первого издания книги.

По сути, изменения в основном коснулись применения различных методов моделирования. Во втором издании мы добавили описание UML2, последнего стандарта одобренного OMG. Во втором издании мы также уделили больше внимания взаимосвязи между управлением требованиями и моделированием, поскольку эта тема достаточно тесно связана с парадигмой метода расширенных связей.

Глава, посвященная средству управления требованиями с помощью Telelogic DOORS, была обновлена и содержит примеры использования последней версии DOORS* и DOORS/Analyst.

Хочется особенно подчеркнуть тот факт, что DOORS/Analyst позволят выполнять моделирование в рамках средства управления требованиями, что дает дополнительные преимущества при разработке.

Книга по-прежнему предназначается для студентов и специалистов, занимающихся проектированием систем, желающих расширить свои знания в области управления требованиями.

Как и ранее, на сайте http://www.requirementsengineering.info можно найти дополнительные материалы о данной книге.

Элизабет Халл
Кен Джексон
Джереми Дик

июнь 2004

* *прим. переводчика.* На момент выхода в свет английского издания книги доступной версией Telelogic DOORS была версия 7. При переводе книги на русский язык было принято решение использовать для примеров последнюю существующую на момент перевода версию, а именно – DOORS 8.0.

Предисловие к первому изданию

Управление требованиями (*Requirements Engineering*) является широко распространенным и привычным термином, однако, не смотря на это, для большей части людей он до сих пор остается непонятным; другая же часть и вовсе понимает его неправильно.

Это уже само по себе является причиной того, что требования бывают очень часто плохо написаны, а значит и плохо управляются. Возрастающий на организацию прессинг современных условий существования и конкуренции является основной причиной отказа ее руководства от внедрения более четких процедур в области управления требованиями. При этом совершенно упускается из виду то, что управление требованиями может помочь организации работать более эффективно. Задача аналитика (проектировщика) помочь своей организации установить такой процесс управления требованиями, который максимально подходит для достижения целей, стоящих перед организацией.

В настоящее время проектирование систем является жизненно важной составляющей экономики, а управление требованиями является одной из важнейших стадий системного проектирования. Хорошо отлаженный процесс системного проектирования является основой для управления требованиями, он определяет эффективность и скорость выхода продукта на рынок. Это особенно важно для тех отраслей, где время выхода продукта на рынок и степень удовлетворения требований заказчика являются ключевыми факторами успеха.

В управлении требованиями присутствует значительная организационная составляющая, которая помогает с помощью требований управлять не только стадией проектирования, но и всем проектом в общем и целом.

В книге уделяется особое внимание тому, как писать хорошие требования. Как основа для более глубокого понимания сути процесса разработки требований, в книге вводиться понятие общего процесса разработки требований. Применение общего процесса рассматривается затем отдельно для области проблем и для области решений.

Книга также описывает основные широко распространенные методы моделирования. В книге подробно рассматриваются связи, их характеристики и методы их анализа. В книге также дается описание программного пакета Telelogic DOORS, предназначенного для управления требованиями. На примерах достаточно подробно демонстрируется использование возможностей DOORS для реализации основных подходов, рассмотренных в книге.

Книга предназначена практикам, занимающимся проектированием систем и желающим углубить свои знания в области управления требованиями. Книга также может быть интересна и студентам последних курсов, специальность которых связана с компьютерными дисциплинами и разработкой программного обеспечения и которые изучают курс разработки требований, а также аспирантам, занимающимся проблемами в области информатики или системотехники.

Подход, изложенный в книге, основан на научных исследованиях в области разработки требований, однако, несмотря на это, книга отражает не только академическую точку зрения, но она также базируется на большом промышленном опыте, что позволяет

специалистам взять практические идеи на вооружение для того, чтобы лучше управлять требованиями (и проектами).

Книга представляет собой отражение всего самого передового и лучшего, что есть на текущий момент в области управления требованиями.

На сайте http://www.requirementsengineering.info можно найти дополнительные материалы.

Элизабет Халл
Кен Джексон
Джереми Дик

май 2002

Благодарности

Авторы выражают большую благодарность отдельным людям и организациям, оказавших помощь в создании книги:

Ричарду Стивенсу, который вдохновил нас на работу в области управления требованиями и заложил фундамент работы над книгой. Он основал QSS Ltd, которая разработала пакет DOORS.

Лесу Оливеру из Astrium за помощь в разработке диаграмм состояний для согласования, тестирования и удовлетворения.

Praxis Critical Systems Ltd за первоначальную концепцию проверки спецификаций, которая в последствии переросла в концепцию *расширенных связей.*

Кейту Коллейру, Джилу Барнетту и другим коллегам из Telelogic Ltd за идеи, которые нашли отражение в книге, за комментарии, советы и поддержку.

Содержание

1 Введение

Когда человек не знает к какой пристани он держит путь,
для него никакой ветер не будет попутным.
Сенека Луций Анней (Lucius Annaeus Seneca)
философ, (65-3 до н.э.)

1.1 Введение в требования

Сегодня, как никогда, проекты по разработке систем[1] нуждаются в «попутном ветре». Быстро меняющиеся технологии и увеличивающаяся конкуренция оказывают большое давление на процессы разработки. Эффективное управление требованиями лежит в основе способности организации крепко держать в руках штурвал и удерживать корабль на плаву в нарастающем потоке сложности.

В настоящее время программное обеспечение является преобладающей побудительной причиной появления новых продуктов. Эта тенденция обусловлена тремя ключевыми факторами:

Возможность построения сколь угодно сложных систем.
Наиболее сложные системы имеют тенденцию содержать в себе программное обеспечение, зачастую очень глубоко интегрированное в компоненты системы. Сложность таких продуктов ограничена только воображением.

Быстрое распространение.
Сегодня компания может задумать новый продукт, разработать его в виде программного обеспечения и очень быстро распространить его по всему миру. Например, производитель легковых автомобилей может улучшить программное обеспечение для своих систем диагностики и распространить его по всему миру десяткам тысяч автомобильных дилеров в течение одного дня.

Готовые ("Off-the-shelf") модули.
В настоящее время системы могут собираться из готовых модулей, приобретаемых вместе с технологиями, что значительно сокращает цикл разработки продукта.

Эти тенденции дают возможность монополизировать все выгоды от разработки новой технологии или продукта за счет быстрого выполнения проекта без привлечения больших производственных мощностей.

В таких условиях появляется острая необходимость в сокращении цикла разработки и времени выхода на рынок ("time to market"). Однако только лишь этот показатель - время выхода на рынок - не дает достаточного преимущества. Стратегической целью становится время выхода на рынок с «правильным» продуктом ("time to market with the right product").

Требования это как раз то, что позволяет нам получить наглядное и согласованное представление о «правильном» продукте. Жизненно важная часть проектирования любых

[1] Под системой понимается любая инженерная система, не только разработка программного обеспечения.

систем это разработка требований, которая, в первую очередь, определяет саму проблемную область, а затем последовательно соотносит все последующие технические решения с конкретными проблемами из этой проблемной области. Действуя только этим путем, можно ожидать, что в результате разработки получится решение, которое будет нас полностью удовлетворять и которое, что немаловажно, - будет рентабельно.

Требования являются основой для любого проекта. Они определяют те потребности «заинтересованных сторон» (stakeholders) – пользователей, потребителей, поставщиков, разработчиков и самого бизнеса, - которые являются для них необходимыми, а также тот функционал, которым система должна впоследствии обладать, чтобы удовлетворить эти потребности.

Для того чтобы стать всем понятными, требования в большинстве случаев пишутся на «обычном» языке, что привносит проблемы другого рода: необходимость полностью и однозначно обозначить проблемы и зафиксировать потребности без использования профессионального жаргона или предварительных договоренностей – является весьма сложной задачей.

Только согласованные требования могут быть основой для проекта, однако, с течением времени потребностей у заинтересованных сторон может становиться все больше и больше и это притом, что интересы сторон могут вступать в конфликт межу собой. Кроме того, потребности могут быть нечетко выражены в начале проекта, их удовлетворение может быть ограничено факторами, лежащими в неконтролируемой области, на удовлетворение потребностей могут влиять другие цели проекта, которые, в свою очередь, тоже могут изменяться с течением времени.

Таким образом, без относительно стабильных базовых и согласованных требований проект будет только «барахтаться на волнах». Это как отправляться в путешествие по морю, не зная конечного места назначения и не взяв с собой навигационные карты. Поэтому требования обеспечивают и «навигационные карты», и средства управления кораблем, помогая достигнуть выбранного пункта назначения.

Согласованные требования обеспечивают базу для планирования разработки системы и ее приемки по завершении работ. Требования необходимы, когда приходится идти на компромиссы, и они жизненно важны, когда в процессе разработки приходиться вносить изменения, что фактически неизбежно для любого из проектов. Как можно дать оценку влияния предлагаемого изменения без детальной проработки модели системы? Или с другой стороны, - а что будет, если отменить внесенные ранее изменения?

Даже если проблема и ее решение определены, необходимо оценить риски, которые могут привести к провалу проекта. Редкий заказчик будет поддерживать проект без убедительной стратегии управления рисками. Организация работы с требованиями позволяет управлять рисками на самых ранних стадиях разработки. Так риск, вытекающий из определенного требования, может быть отслежен, может быть проведена оценка его влияния, вероятность его появления (реализации), и, как следствие, может быть разработан предварительный план по предотвращению и устранению последствий этого риска. И что самое главное, - все это можно выполнить задолго до того, как возникнет необходимость соответствующих затрат.

Следовательно, требования являются базой для:

- планирования проекта;
- управления рисками;
- приемочного тестирования;

- компромиссов (согласований);
- управления изменениями.

Наиболее распространенные проблемы, из-за которых «проваливаются» проекты отнюдь не технические.

В таблице 1.1 приводятся основные причины провалов проектов. Таблица заполнена на основании данных опроса, проведенного Стэндиш Групп (Standish Group) в 1995-96 годах, и показывает процентное соотношение причин, приведших к «смерти» проектов. Точками отмечены причины связанные непосредственно с требованиями.

Перечисленные проблемы можно разделить на три основных категории:

- Требования – плохо организованы, плохо написаны; слабо связаны с запросами и потребностями заинтересованных сторон (stakeholders); очень быстро изменяются, или изменяются без необходимости; нереалистические ожидания.
- Проблемы, связанные с недостатком ресурсов – недостаток денег, недостаточная поддержка или даже полный провал в установлении необходимой дисциплины планирования; многие из этих факторов также являются следствием плохого управления требованиями.
- Искусство управления – выражается во влиянии на первые две категории.

Все эти проблемы могут быть решены за относительно небольшую цену.

Таблица 1.1 Причины провалов проектов

•	Неполнота требований	13.1%
•	Недостаточное привлечение пользователей	12.4%
	Недостаток в ресурсах	10.6%
•	Нереалистические ожидания	9.9%
	Недостаток поддержки от руководства	9.3%
	Изменение требований/спецификаций	8.7%
	Недостаточное планирование	8.1%
	Потеря необходимости	7.5%

Источники: Standish Group, 1995 и 1996; *Scientific American,* September 1994

Таблица 1.2 Факторы успехов проектов

•	Вовлечение пользователей	15.9%
	Поддержка руководства	13.9%
•	Четкая и ясная постановка требований	13.0%
	Хорошее планирование	9.6%
•	Реалистичные ожидания	8.2%
	Частые контрольные точки	7.7%
	Компетентная команда	7.2%
	Владение (требованиями)	5.3%

Источники: Standish Group, 1995 и 1996; *Scientific American,* September 1994

Факторы, способствующие успеху проектов, приведенные в таблице 1.2, не являются прямой противоположностью причин провальных проектов.

Постоянная поддержка руководства и правильное планирование являются очень важными факторами: чем больше проект по продолжительности и по объему, тем больше шансов провалить его (*Scientific American*, September 1994).

В этой книге рассматриваются методы разработки требований, в общем, и управления требованиями, в частности.

В книге поясняется, в чем состоят различия между требованиями заинтересованных сторон (stakeholders requirements)[2] и системными требованиями. Объясняется, как применять требования для управления разработкой.

В книге продемонстрировано как организация связей (traceability) - от пользовательских требований (stakeholder requirements) через системные требования (system requirements) к спецификациям системы (design) - и контроль за ними могут быть использованы для оценки развития проекта, управления изменениями и определения рисков.

Общаясь с книгой, читатель раскроет для себя аспекты, связанные с верификацией требований и тестируемости компонентов системы, которые проектировались и создавались в соответствии с этими требованиями, а также то, как формулировать запросы на тестирование и приемочные испытания.

Книга обратит ваше внимание на необходимость организации работ по проектированию (создания спецификаций (designs)) таким образом, чтобы сделать легкой последующие тестирование и интеграцию компонентов системы.

Присутствие в книге Главы 8, «Аспекты управления разработкой требований» обусловлено важностью прослеживающейся связи между управлением требованиями и управлением проектом.

1.2 Введение в системное проектирование

Эта книга посвящена не только требованиям для программного обеспечения (ПО). Принципы и практические методы разработки требований применимы и для системных разработок, в которых ПО может являться лишь небольшой частью всей системы.

В качестве примера можно рассмотреть железнодорожную систему – Магистраль Западного Побережья (West Coast Mainline) от Лондона до Глазго.

Требование высокого уровня для системы могло бы звучать так: «поездка от станции Юстон (сев. Лондон) до Глазго (Шотландия), должна занимать не более 250 минут».

Удовлетворение этого требования зависит от скоординированного взаимодействия всех основных составляющих системы:

- поезда и скорость их движения;
- рельсы и их способность выдерживать высокоскоростные поезда;
- станции и их персонал, а также время ожидания, которое накладывается на станционное расписание поездов;
- машинисты и их способность управлять поездами;
- системы сигнализации и связи;

[2] В дальнейшем изложении требования заинтересованных сторон (stakeholders requirements) будут называться требованиями пользователей или пользовательскими требованиями.

- системы управления железнодорожным движением;
- службы энергообеспечения.

Несмотря на то, что системы сигнализации и связи, а также системы управления железнодорожным движением играют крайне важную роль для выполнения этого требования, они не могут обеспечить его выполнение сами, т.е. без участия всех остальных составляющих системы. Полное решение задачи требует использования всех компонентов системы. Так и в реальной жизни - большинство требований удовлетворяется набором свойств системы, поскольку являются результатом поведения системы как единого целого.

Что же мы тогда подразумеваем под системой?

Система это:

- набор компонентов – механизмов, программного обеспечения и людей, -
- которые согласовано взаимодействуют
- для достижения некоторого заданного результата, сформулированного в виде требований.

Таким образом, системы включают людей.

Для Магистрали Западного Побережья машинисты и станционный персонал, включая обучение, которое они проходят и инструкции, которыми они пользуются, – важны также, как механизмы и ПО.

Вследствие того, что компоненты системы должны взаимодействовать друг с другом, интерфейсы между ними занимают важное место в системном проектировании (и разработке требований) – интерфейсы между людьми и механизмами, между механизмами и ПО.

В качестве примера интерфейса между двумя механизмами для железнодорожной системы можно рассмотреть взаимодействие колес поезда с рельсами. Помимо физического расположения (система спроектирована так, чтобы поезд шел по рельсам без схождения с них), существующие электрические цепи, расположенные параллельно рельсам, могут также рассматриваться как часть системы управления железнодорожным движением - для определения положения поезда.

В центре концепции «системы» лежит идея «системных свойств» (emergent properties), поскольку полезность системы не зависит от какой-то конкретной ее части, а появляется в результате взаимодействия между ее компонентами. Системные свойства могут быть желательными, такими, какими они были изначально задуманы и спроектированы, для того, чтобы сделать систему полезной; или - нежелательными, другими словами, могут являться непредвиденными побочными эффектами, такими, например, как загрязнение окружающей среды. Искусство системного проектирования состоит в том, чтобы получать желательные системные свойства и избегать нежелательных.

Другая важная концепция это «система систем».

Любая система может быть разработана так, чтобы, в свою очередь, являться составной частью большей системы. Например, Магистраль Западного Побережья является частью большей железнодорожной системы и пересекается с другими более и менее важным дорогами. Сама же железнодорожная система это тоже часть большей транспортной системы страны, которая взаимодействует со всеми типами дорожных систем и системами воздушного транспорта. Транспортная система, в свою очередь, обеспечивает

инфраструктуру для транспортировки людей и грузов, и, таким образом, является частью экономики страны. Страна уже может рассматриваться как часть мира. И так далее.

Для того чтобы правильно сформулировать требования к системе, необходимо понять требования к той системе, в чей состав она входит. Зачастую, от этого зависит правильное функционирование всей системы. Например, способность вертолета летать обеспечивается наличием у Земли гравитации и атмосферы.

Возьмем другой, очень простой пример – чашку (рис. 1.1).

Рис.1.1 Чашка как пример простой системы.

Очевидно, что чашка состоит из компонентов: ручки и чаши. Для каких целей нужны эти компоненты? - чаша нужна для того, чтобы содержать в себе жидкость, а ручка нужна для того, чтобы человек мог держать чашку, не обжигаясь. Отсюда вывод – цель, или требования для чашки, - это позволить человеку донести горячую жидкость до рта, не разлив ее и не получив ожога.

Чашка полна интерфейсов. Она может быть поставлена на плоскую поверхность для устойчивости; ее можно держать в человеческой руке; она может быть наполнена жидкостью и опустошена; должен быть интерфейс с жидкостью для продолжительного ее содержания; и она должна подходить для перемещения жидкости в человеческий рот.

Тем не менее, можно сделать и другие наблюдения:

- Сама по себе чашка не может удовлетворить цель, сформулированную нами выше, так как чашка зависит от движений человеческой руки, которая ее держит.

- Чаша, как компонент чашки, критически зависит от гравитации, а также правильного использования. Если чашку перевернуть вверх дном, это приведет к выливанию жидкости, и, возможно, ожогу.

Итак, способность простой чашки удовлетворить сформулированную нами цель зависит от:

- свойств, которые появились в результате взаимодействия ее компонентов;
- соответствующих интерфейсов с внешними компонентами;
- ее корректного включения во общую систему – чашка удерживается и переносится человеческой рукой;
- присутствия соответствующих внешних условий – в условиях невесомости для удовлетворения поставленной нами цели явно потребуется другое решение.

Резюмируя вышесказанное, необходимо отметить, что разработка требований должна обязательно принимать во внимание природу системы. Нельзя рассматривать системные свойства изолированно, - необходим комплексный подход: учет условий, которые привносит внешнее окружение, накладываемые кем-то или чем-то ограничения, а также интерфейсы с окружающими системами.

1.3 Требования и качество

Результаты отсутствия требований множественны и различны.
Вокруг нас достаточно свидетельств того, что системы получались неудачными только из-за того, что требования не были правильно организованы. И даже более того, система может быть создана и даже функционировать, но если она не будет делать то, что от нее хотят пользователи, она будет бесполезна.

Очень интересно рассмотреть связь между требованиями и качеством.
Термин «качество» может пониматься по-разному. Если заговорить о качественном легковом автомобиле, то кто-то может упомянуть Роллс-ройс, Мерседес или Ягуар. И это будет пример старой и известной путаницы между «качеством» и «роскошью». Такой подход к формулировке качества и такая его оценка рассыплются как карточный домик, если вы начнете выбирать качественный автомобиль для ралли. Никто не выбирает себе ни Роллс-ройс, ни Мерседес, ни Ягуар, поскольку они не обладают правильными характеристиками для данного случая - соотношения мощности к весу, клиренса, прочности и т.д. Недавние исследования, например, показали, что наиболее качественным автомобилем (в своем классе) может быть названа Шкода – не самый роскошный автомобиль, но зато имеющий набор соответствующих качеств.

Таким образом, «качество» есть соответствие системы целям или требованиям – это обеспечение того, что удовлетворяет потребителя и в тоже время гарантирует, что нужды всех заинтересованных сторон учтены.

Как мы увидим в Главе 8, работа с требованиями является дополнением к другим условиям, которые руководитель проекта должен непременно учитывать, например, стоимость и график реализации, чтобы обеспечить достижение необходимого качества. Каждое управленческое решение является неким компромиссом между стоимостью, сроками работ и качеством, которые являются тремя взаимосвязанными осями координат.

Поскольку работа с требованиями это дисциплина, которая начинает применяться с самого раннего этапа жизненного цикла разработки, то ее влияние на качество конечного результата, которое может быть достигнуто путем надлежащего управления требованиями, увеличивается прямо пропорционально продвижению по этапам жизненного цикла. Относительно небольшие усилия, затраченные на ранних стадиях разработки, могут принести огромную пользу на последующих стадиях.
Изречение «качество бесплатно» (заголовок книги Фила Кросби (Phil Crosby)) является правдивым лишь в том смысле, что правильное начало может сохранить на более поздних этапах невероятное количество усилий, которые возможно могли бы понадобится на исправление уже сложившегося положения вещей, если бы вначале все пошло неправильно.
Улучшение требований означает улучшение качества продукта.

1.4 Требования и процесс выполнения проекта

Существует распространенное заблуждение, что разработка требований это всего лишь единичная фаза, которая выполняется и заканчивается в начале разработки продукта.
Цель данного раздела показать, что разработка требований - как дисциплина - играет жизненно важную роль на каждом этапе проекта.

Вначале давайте рассмотрим самую последнюю фазу проекта – приемочные испытания. На вопрос «в соответствии с чем должны выполняться приемочные испытания» напрашивается ответ – «в соответствии с пользовательскими требованиями (stakeholder requirements)». Сразу становиться очевидным, что требования, разработанные в самом начале проекта, так или иначе, используются на его самом последнем этапе.

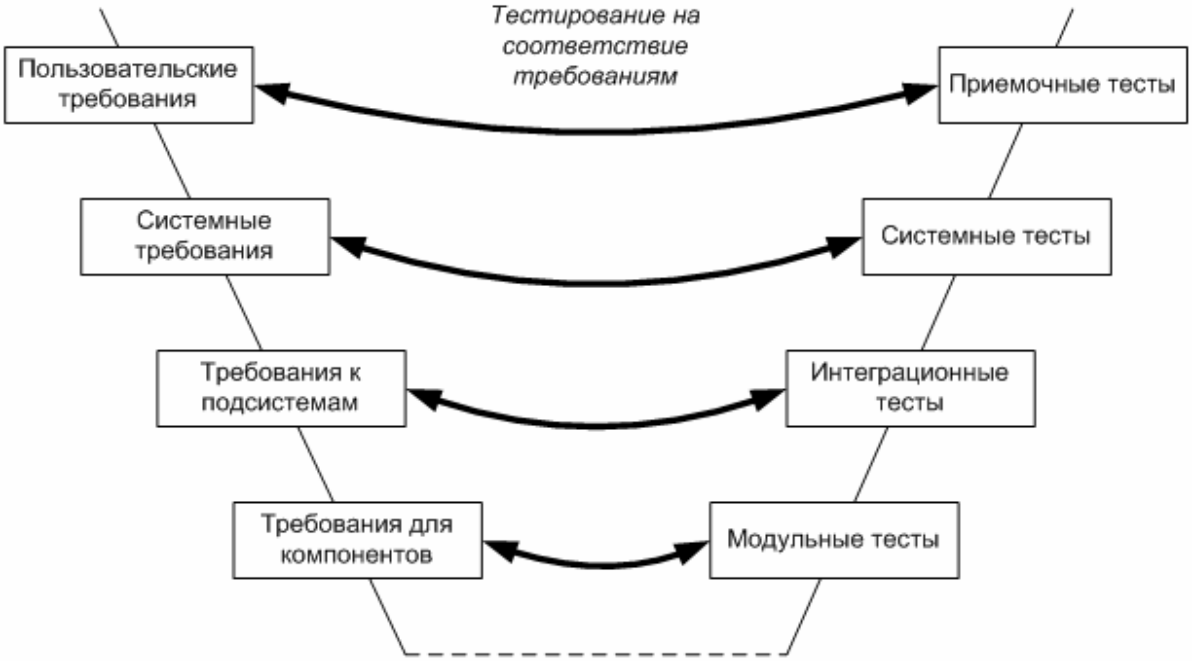


Рис. 1.2 Требования в V-модели.

Классическая V-модель[3] (V-model), описывающая различные стадии проекта, в основном базируется на связи между требованиями и их тестированием.
Рис. 1.2 показывает связь требований и тестирования на каждом этапе проекта.

V-модель также отображает системную разработку в терминах уровней (layers), где каждый уровень соотноситься с определенным этапом разработки. Несмотря на то, что на каждом уровне могут быть использованы несколько различных процессов, основной принцип работы с требованиями не меняется.
Этот вопрос будет рассмотрен подробнее в Главе 2 по ходу введения в общий процесс разработки. Рис. 1.3 демонстрирует цель работы с требованиями на каждом уровне.

Рис.1.3 Уровни разработки требований.

Другая роль, которую могут играть требования в организации, это выстуать в роли средства связи между проектами. Это очень хорошая идея, потому что многие организации желают:

- максимально увеличить использование наработок в разных проектах;
- управлять семействами сходных продуктов;
- использовать программное управление для согласования действий;
- оптимизировать разработку, используя опыт предыдущих проектов.

Хороший набор пользовательских требований (stakeholder requirements) может обеспечить краткое и не техническое описание того, что будет разработано, на уровне, который доступен для понимания высокого руководства.

[3] Произносится как «Ви-модель»

Аналогичным образом, системные требования могут представлять собой прекрасное техническое описание проекта.
Два этих шага служат основой для всех последующих действий в рамках проекта, что и проиллюстрировано на рис. 1.4.

Рис 1.4 Разработка требований в компании

Поскольку требования играют такую важную роль в разработке систем, то с ними необходимо постоянно работать.
Если, например, изменить технические спецификации (design) системы без одновременного и соответствующего изменения всех требований более высокого уровня, то это может привести к огромным проблемам в дальнейшем. Таким образом, разработка требований и управление их изменениями тесно связаны между собой.

Если при работе над проектом возникает необходимость внести изменения (например, возникла техническая проблема, связанная с деталями технического проектирования, или, заказчик настаивает на корректировке первичных формулировок), то должно быть обязательно учтено влияние такого изменения на качество, стоимость и график работ.

При внесении изменения необходимо в основном выполнить следующее:

- принять или отклонить изменение;
- согласовать стоимость изменения с заказчиками/поставщиками;
- организовать работы по переделке.

Основной концепцией, позволяющей проводить анализ влияний такого рода, является возможность установления и последующего контроля связей между требованиями

(requirements traceability). Этот метод рассмотрен более подробно в разделе 1.5, а также в Главах 2 и 7.
Таким образом, можно утверждать, что управление изменениями (change management) является важной частью процесса работы с требованиями. Это хорошо проиллюстрировано на рис. 1.5.

Рис.1.5 Риски управления изменениями связанные с взаимосвязанностью требований

Становится очевидным, что совершенно независимо от управления изменениями, способность руководителя управлять проектом в целом в значительной степени зависит от хорошего поставленного процесса работы с требованиями.
Без требований, руководитель проекта не имеет средств оценки того, на сколько хорошо идет работа над проектом и в правильном ли направлении все это движется. Если требований нет как таковых, то в тот момент, когда нужно сделать изменение, не будет той базовой точки, относительно которой можно будет оценивать влияние этого изменения на ход проекта. Более того, когда в такой ситуации – при отсутствии требований - возникают изменения, то все что может сделать руководитель проекта, это лишь вмешиваться в технические детали, что недопустимо для его роли в проекте, поскольку решения на техническом уровне должны принимать технические эксперты – инженеры.
Требования, хорошо сформулированные на каждом из соответствующих уровней, дают руководителю проекта правильное представление о проекте, о ходе его выполнения и дают ему шанс правильно выполнять свою роль.

Таким образом, требования очень важны для любого системной разработки.
Они влияют на весь процесс разработки от самого начала до его полного завершения. Без эффективного процесса управления требованиями, процесс разработки системы будет выглядеть как корабль, который во время шторма дрейфует без управления.

Более того, только хорошее управление требованиями, с учетом мнения заказчиков и потребителей, дает компании основу для плодотворного общения и взаимодействия в процессе выполнения проекта.

1.5 Создание и анализ связей между требованиями

В контексте разработки требований, создание и анализ связей необходимы, в первую очередь, для понимания того, как требования высокого уровня – общие цели, задачи, пожелания, предполагаемые ожидания, нужды и т.п. - трансформируются в требования низкого уровня. Следовательно, в основном, связи нужны между различными уровнями информации.

В контексте бизнеса кому-то может быть интересно знать как:

- стратегия бизнеса
  конкретизируется как
- задачи бизнеса,
  которые воплощаются как
- организация бизнеса и бизнес-процессов.

В контексте системного проектирования интерес может фокусироваться на том, как:

- пользовательские требования (stakeholder requirements)
  удовлетворяются
- системными требованиями,
  которые разделяются на
- подсистемы,
  которые реализуются как
- компоненты.

Использование связей может принести следующие выгоды:

- *Бо**льшая уверенность в достижении целей.*** Установление связей и формализация их контроля приводит к четкому пониманию того, как именно достигаются цели.
- *Возможность оценить влияние изменений.* Существование связей между требованиями дает возможность проводить разного рода анализ влияния вносимых изменений.
- *Возможность оценить вклад подрядчиков и субподрядчиков.* Появляется возможность ясно оценить ту часть работы, которую выполняют по проекту другие организации.
- *Возможность контролировать ход проекта и оценивать объем выполненной работы.* Обычно бывает очень трудно оценить выполненную вами работу как часть общего объема работ по проекту, тем более, если сама работа заключается в написании и редактировании документов. Использование связей позволяет достаточно точно измерять прогресс даже на ранних этапах проекта.
- *Возможность сопоставлять затраты и возможную выгоду (определять экономическую целесообразность).* Однозначное определение связи между

требованиями и определенными компонентами системы, позволяет соизмерять затраты с предполагаемым положительным эффектом от их реализации.

Связи между требованиями обычно имеют тип «многие ко многим». Одно требование нижнего уровня может быть связано с несколькими требованиями более высокого уровня, и наоборот. В простейшем же случае связи могут устанавливаться между разными требованиями, но одного уровня.

Программные средства поддержки работы с требованиями обычно позволяют создавать связи путем «перетаскивания одного параграфа документа на другой» (“drag-n-drop”). Связи между требования выглядят как гиперссылки на web страницах, но в отличие от них должны иметь возможность прослеживаться в любом направлении. На рис. 1.6. показано, например, как выглядят связи между различными уровнями требований, а также связи и между требованиями и соответствующими тестами.

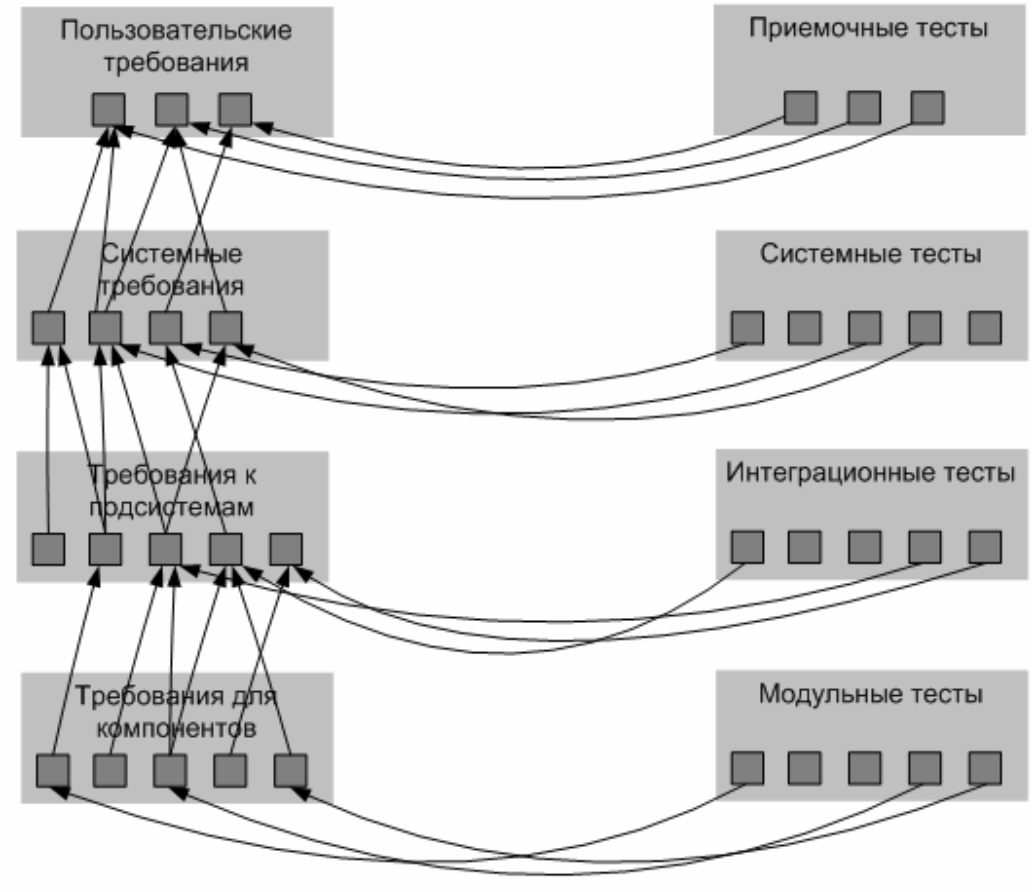


Рис. 1.6 Связи между требованиями

Стрелки на линиях связях проставляются исходя из конкретного правила - стрелка всегда указывает направление к источнику информации. Другими словами, если одна информация «вытекает» из другой, то связь должна быть направлена от первого ко второму. Для такого соглашения есть две причины:

- Такой формат стрелки зачастую соответствует хронологическому порядку появления информации - связь всегда указывает на информацию, которая существовала ранее.
- Очень часто это соответствует также и правам на владение информацией: одному человеку принадлежат исходящие из документа связи, другому – только входящие.

Для поддержки процесса работы с требованиями используются различные методы анализа связей между требованиями (см. таблицу 1.3).

Таблица 1.3 Методы анализа связей требований

Метод анализа	Описание	Поддерживаемый процесс
Анализ влияния	Анализ входящих связей с целью ответа на вопрос: «Что будет если изменить это требование?»	Процесс изменения
Анализ последствий	Анализ исходящих связей с целью ответа на вопрос: «Нам это действительно нужно?»	Анализ экономической целесообразности
Анализ покрытия	Анализ связей с целью ответа на вопрос: «Все ли учтено?» Обычно используется для оценки прогресса работы.	Проектирование. Отчетность руководству

Анализ влияния дает возможность оценить, на какие другие элементы проекта (нижнего уровня) повлияет внесение изменения в данный конкретный элемент (см. рис. 1.7).
А поскольку зачастую влияние одного элемента на другие носит относительный характер, поэтому каждый раз при внесении изменения, если это необходимо, проводить более детальный анализ того, затронет ли данное изменение другие элементы проекта или нет и, если затронет, то насколько сильно.

Анализ последствий работает в противоположном направлении анализу связей. Анализируются элементы нижнего уровня, например, содержание требования, часть спецификации или результат теста, а затем с помощью связей проверяется его соответствие одному из элементов более высокого уровня.
Элементы нижних уровней, если они не имеют никаких исходящих «наверх» связей, вероятно, лишь увеличивают затраты и не приносят никакой пользы.

Анализ покрытия используется для анализа входящих связей, для того чтобы проверить, что все требования высокого уровня связаны со спецификациями на более низких уровнях и тестами. Отсутствие таких связей свидетельствует о том, что требование не будет удовлетворено (выполнено) или протестировано. Наличие связей, само по себе, не дает гарантии, что требование будет удовлетворено и протестировано. Для того, чтобы убедиться в этом необходимо приложить талант, знания и опыт проектировщика системы.

Анализ покрытия также применяется для того, чтобы измерить прогресс работы - то, насколько системные инженеры продвинулась в удовлетворении требований пользователей. Подразумевается, что системные инженеры разрабатывают системные требования, отвечающие требованиям пользователей, или, как еще говорят, «покрывающие» их.

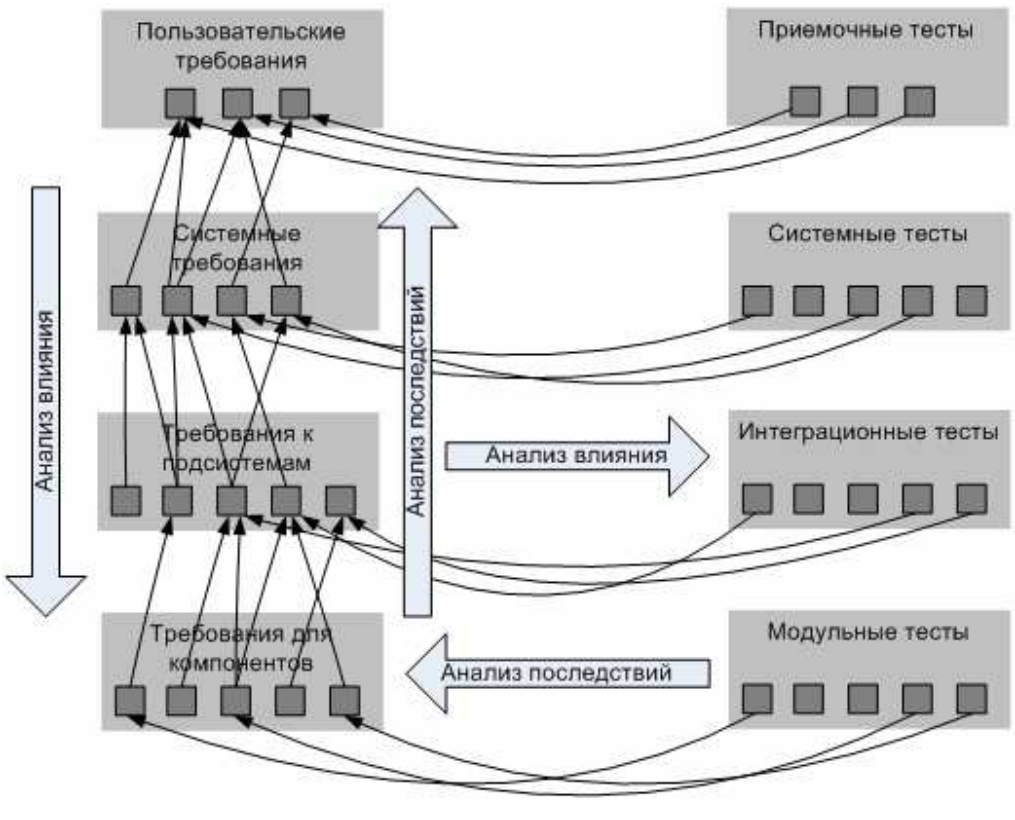


Рис. 1.7 Анализ влияния и анализ последствий

В процессе разработки, каждое системное требование связывается с теми пользовательскими требованиями, для удовлетворения которых оно и предназначено. (Следует заметить следующее - если установка связей происходит непосредственно в процессе написания системных требований, то это требует совсем небольших дополнительных затрат по сравнению с тем случаем, когда установлением связей занимаются уже после того, как и пользовательские и системные требования уже написаны!)

На любом этапе разработки системных требований процент их готовности может быть определен как процент покрытия пользовательских требований системными. Это очень удобно для руководящего персонала, поскольку в любой момент времени дает возможность контролировать прогресс (процесс развития событий) даже на самых ранних стадиях разработки.

Тот же самый принцип может быть использован для измерения прогресса разработки планов тестирования. Процент готовых планов тестирования может быть определен как процент покрытия требований тестами.

Использование анализа покрытия в этих двух измерениях проиллюстрировано на рис. 1.8.

Рассмотренные типы анализа связей позволяют сделать вывод о том, что связи являются, с одной стороны, - очень простой, а с другой стороны, - ключевой концепцией процесса работы с требованиями.

Более подробно связи и их анализ рассмотрены в Главе 7.

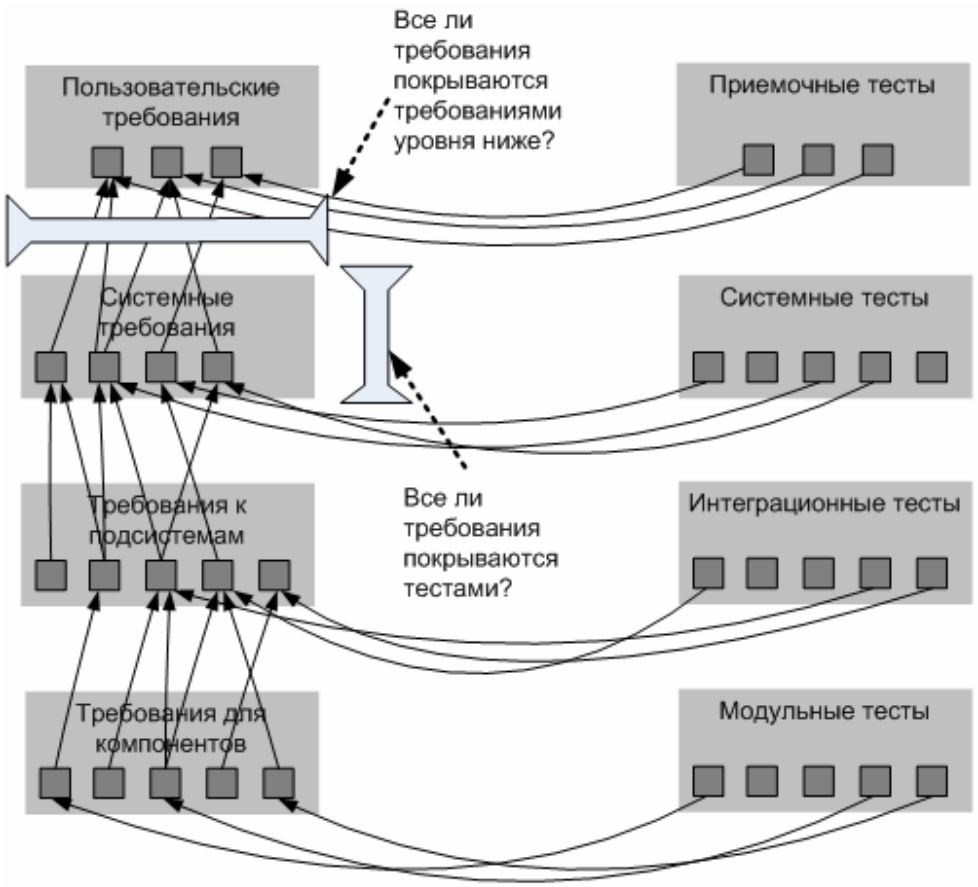


Рис 1.8 Анализ покрытия

1.6 Разработка требований и моделирование

Очень важно видеть и понимать взаимосвязь между разработкой требований и системным моделированием. Хотя это и взаимодополняющие дисциплины, но они, тем не менее, не должны восприниматься как одно и тоже.

Рис. 1.9 иллюстрирует взаимосвязь моделирования и разработки требований на примере сэндвича. В этой метафоре отражается «хлеб с маслом»[4] процесса разработки. Системное моделирование является «начинкой» - между соседними уровнями требований, - которая необходима для перехода от верхнего уровня к нижнему. Эта «начинка» состоит из анализа и реализации, т.е. выработки конкретных решений.

Некоторые употребляют термин «моделирование требований». Это термин является неверным по сути. Моделируется реализация системы, а не требования к ней. Моделирование поддерживает наиболее творческий процесс – разработку реализации системы, выработку конкретных системных решений. Моделирование помогает инженеру уточнять и детализировать реализацию системы путем разбиения ее на компоненты при

[4] Средства к существованию. От выражения: «зарабатывать на хлеб с маслом»

движении вниз от одного уровня требований к другому. Требования это полный «снимок» того, что именно требуется от системы на каждом уровне. При этом детализация «снимка» увеличивается при движении вниз от уровня к уровню.

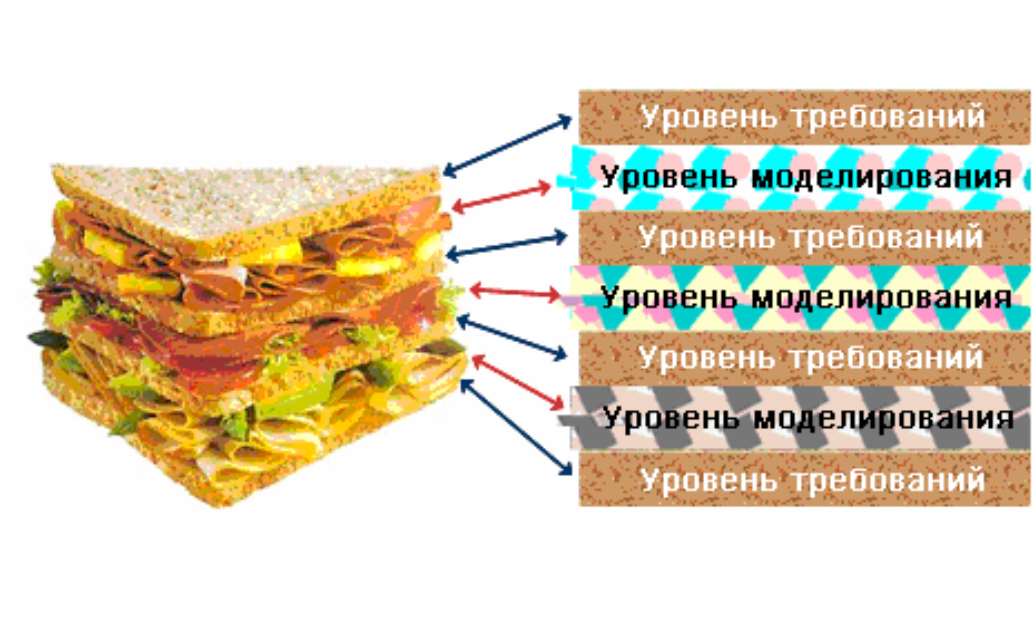


Рис 1.9 Сэндвич системной инженерии

Определенная модель никогда не описывает систему полностью, а если и описывает, то это уже не модель. Поэтому для описания различных аспектов системы обычно используется несколько различных моделей, возможно даже взаимосвязанных. При этом вовсе не исключено, что некоторые аспекты системы могут быть просто описаны требованиями (в текстуальном виде) и не покрываться моделированием.

Модель это некая абстракция системы, которая сознательно уделяет особое внимание некоторым аспектам системы и вовсе исключает из рассмотрения другие.
Абстракция, по сути, это борьба с рассеиванием внимания – игнорирование тех деталей, которые не важны для данной конкретной модели, не смотря на то, что эти детали, возможно, очень даже важны для системы в целом. Преимуществом такого подхода является то, что небольшой объем связанной информации, относящийся к определенному аспекту системы, может быть собран в одной модели, обработан, структурирован и проанализирован методами, которые наиболее всего подходят для работы с информацией такого типа.

Там, где необходима работа с большими объемами информации, моделирование позволяет собрать определенное подмножество данных, отвечающих одной определенной цели и сосредоточиться на нем, а затем посмотреть шире, чтобы увидеть систему в целом. Возможность сосредоточиться в определенный момент времени только на маленьком объеме информации (определенном аспекте системы) обеспечивает возможность правильной работы со всей системой в целом.

На рис. 1.10 показаны взаимосвязанные роли, которые играют разработка требований и системное моделирование.

Рис 1.10 Требования и моделирование

На определенном уровне моделирование помогает системному инженеру в анализе требований для:

- обсуждения разрабатываемой системы с заказчиком и улучшения взаимопонимания с коллегами;
- анализа системы с целью убедиться в наличии желаемых системных свойств (emergent properties) и, что также немаловажно, - в отсутствии нежелательных свойств;
- понимания того, как будет проверяться реализация данных требований, при трансформации этих требований в новые – на более низком уровне.

Природа моделей на разных уровнях различна. На верхнем уровне используются «пользовательские сценарии» (stakeholder scenarios) для получения пользовательских требований (stakeholder requirements).
Далее, для перехода от пользовательских требований к системным требованиям могут использоваться различные типы функциональных моделей. Так, например, функциональные модели могут включать в себя следующие UML[5]-диаграммы: диаграммы классов (class diagrams), диаграммы последовательности сообщений (message sequence charts) и диаграммы состояний (state charts). (Более подробно данная техника моделирования будет рассмотрена в Главе 3.)

При переходе от системных требований к архитектуре, большее значение начинают играть различные аспекты производительности создаваемой системы. В этом случае несколько различных типов моделей могут быть использованы для того, чтобы продемонстрировать то, что выбранная архитектура может удовлетворить как функциональным, так и нефункциональным требованиям. При этом модели могут быть самыми разными. Для моделирования производительности может применяться теория

[5] UML – Unified Modeling Language (www.uml.org) – универсальный язык моделирования. Методология и нотация проектирования ПО, разработанная OMG-группой (Object Management Group, www.omg.org), в состав которой входит и Telelogic.

очередей. Для моделирования аэродинамических характеристик может быть использована аэродинамическая труба. Для моделирования движения транспорта – моделирование расписания.

Как следует из приведенных примеров, для различных системы применяются различные типы моделей. Тип модели может меняться от приложения к приложению и зависеть от конкретной природы системы. Моделирование расписаний применяется для проектирования железнодорожных систем, но для разработки воздушных судов оно вряд ли применимо, поскольку в данном случае более уместно использование аэродинамического моделирования. (Аэродинамика, несомненно, важна также и для высокоскоростных поездов). Для моделирования систем передачи данных (и голоса) используются диаграммы последовательности сообщений, а для моделирования систем, имеющих сложную структуру данных, применяются модели, содержащие диаграммы «сущность-связь» (entity-relationship diagrams).

Несмотря на то, что существует большое количество разнообразных типов моделей, следует обратить внимание на тот факт, что принципы управления требованиями остаются постоянными для любых систем.

Поскольку данная книга посвящена работе с требованиями, то она также уделяет внимание дисциплинам, связанным с ней, к числу которых относятся и методы системного моделирования.

1.7 Требования и тестирование

Как обсуждалось выше, на каждом уровне, требования тесно связаны с тестированием. В широком смысле, тестирование это любое действие, направленное на выявление и предотвращение дефектов в системе, где дефект это отклонение от требований. Таким образом, сразу необходимо отметить, что в дополнении к классическим методам тестирования, таким как модульное тестирование, тестирование подсистем, системное тестирование, понятие «тестирование», используемое в данной книге, подразумевает также и рецензирование, инспектирование и анализ с использованием моделирования.

Из-за многообразия способов тестирования, для обозначения их всех в книге используется термин *проверка* (*qualification*).

Проверка должна начинаться как можно раньше, так как откладывание тестирования до конца разработки может привести к значительным расходам и последующей необходимости вносить изменения в спецификации и переделывать систему в случае обнаружения даже малейшей ошибки. Проверка начинается на этапе разработки спецификаций системы (design) и состоит в рецензировании требований, инспекции спецификаций и применении различных форм моделирования системы.

На рис. 1.11 показана V-модель цикла разработки и стратегия проверки системы в процессе разработки.

На левой стороне V-модели (горизонтальная часть рисунка) показаны методы проверки, применяемые на ранних этапах.

На правой стороне показаны методы, применяемые на более поздних этапах.

Здесь следует избегать случаев, когда одно пользовательское требование может послужить причиной большого объема различного рода лишних проверок на последующих стадиях разработки.

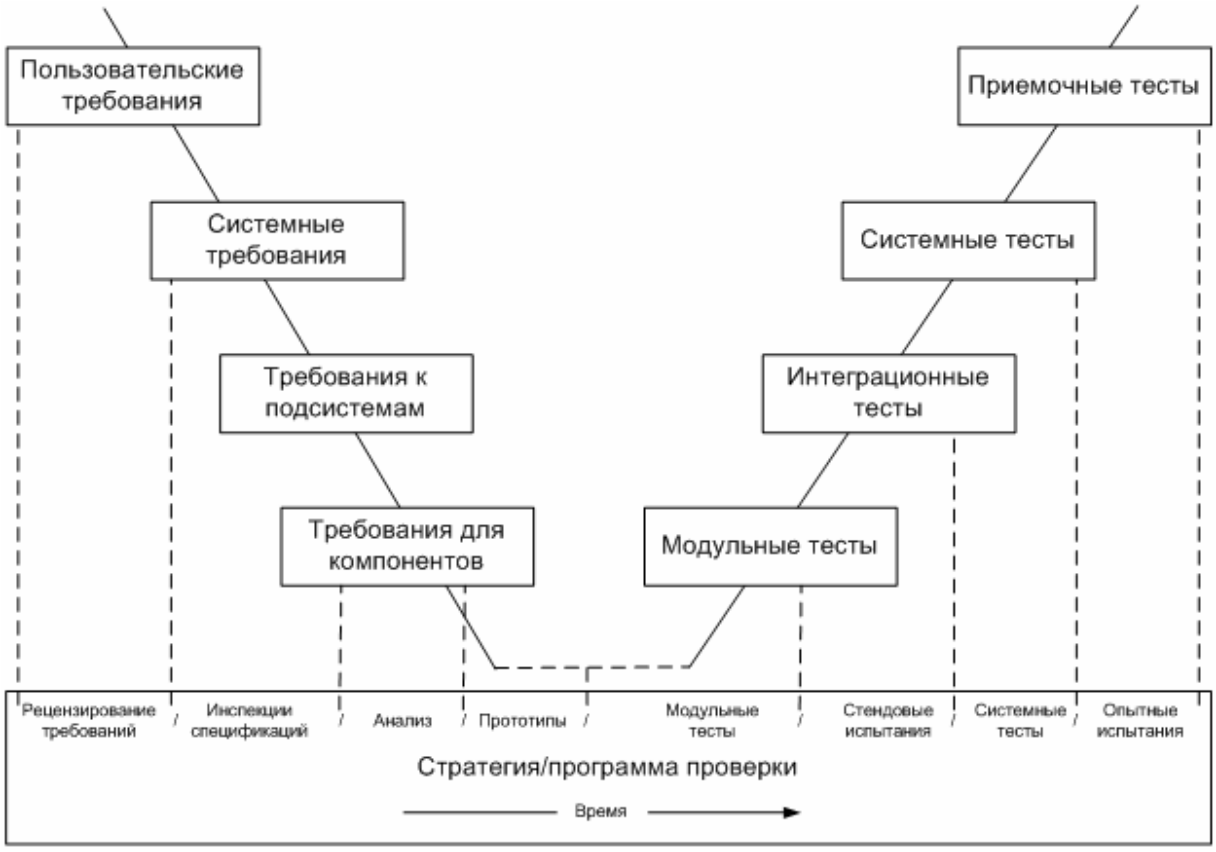


Рис 1.11 Стратегия проверки и V-модель

Если требование считается реализованным, т.е. когда оно удовлетворяется полезными (желаемыми) свойствами на уровне системы, то проведение проверок на уровне составляющих компонентов бывает недостаточно эффективным (по сути, и не нужным), поскольку на этом уровне этих свойств (декларируемых и проверяемых) просто может еще и не быть. Проверка системных свойств должна выполняться на том уровне, на котором эти свойства проявляются в полной мере.

1.8 Требования в области проблем и в области решения

Целью системного проектирования является выработка эффективных решений определенных проблем. Как обсуждалось выше, этот поэтапный процесс является жизненно важным в обеспечении для бизнеса возможности выхода на рынок с «правильным» продуктом в приемлемые сроки и с допустимыми затратами.

В начале процесса разработки самым важным являются требования к разрабатываемому продукту. Как с точки зрения руководства, так и точки зрения системных инженеров необходимо сделать четкое разделение между «областью проблем» и «областью решения». Необходимо, чтобы к проблемной области были отнесены: формулировка проблем, модели использования (прецеденты), пользовательские требования.

Начиная с системных требований, все должно быть, отнесено к области решения.

В таблице 1.4. показано «идеальное» разделение между проблемной областью и областью решения, а также сформулированы цели, которым должны удовлетворять требования трех верхних уровней.

Таблица 1.4 Область проблем и область решения

Уровень требований	Область	Точка зрения	Цель
Пользовательские требования	Область проблем	Пользователь (представитель заинтересованной стороны)	Определяет - ***что*** пользователь желает достичь с помощью создаваемой системы. Следует избегать формулировки конкретных решений.
Системные требования	Область решения	Аналитик	Абстрактно определяет - ***как*** система будет удовлетворять пользовательским требованиям. Следует избегать точных описаний реализации предлагаемых решений.
Системные спецификации (архитектура системы)	Область решения	Архитектор	Определяет - ***как*** конкретная архитектура системы будет удовлетворять системным требованиям.

Здесь необходимо обратить внимание на принцип абстракции, которому нужно следовать, описывая проблемную область.

Первоначальная формулировка возможностей системы должна содержать только то, что необходимо для определения (описания) проблемы, и не должна содержать ничего, что определяет конкретные решения. Это дает свободу системным инженерам для выполнения своей работы, которая как раз и заключается в нахождении наилучшего решения проблемы, что выполняется только при условии, что инженер не связан заранее никакими определенными идеями.

Моделирование, помогая переходить от уровня к уровню, тоже может привносить элементы конкретных решений, уже даже на верхнем уровне. Для того чтобы избежать уклона в плоскость решения, на ранних этапах лучше применять моделирование только для описания включающей системы (подсистемы). Например, при разработке системы связи для судна моделирование, на первых порах, должно быть сосредоточено на описании корабля, а не самих средств связи. Это приводит к формулировке именно проблем в контексте включающей системы, а не описания их возможного решения.

Тот же принцип применим и к системным инженерам (аналитикам), которые должны оставлять свободу архитекторам для выполнения их работы, которая заключается в нахождении лучшего системного решения для абстрактных системных требований. Элементы реализации, создаваемые на этапе функционального моделирования на верхнем уровне требований, не должны содержать деталей, которые будут и должны определяться на последующих стадиях.

Например, в системе управления автомобильным движением:

- Заинтересованные стороны (stakeholders) могут выразить проблему следующим образом: максимизация транспортного потока, при минимизации риска возникновения дорожных происшествий на пересечении дорог при минимизации стоимости обслуживания решения.
- Системные инженеры (system engineers) могут предложить различные решения этой проблемы: разного рода светофоры; организация кругового движения на определенных участках дороги; а может быть и мост в качестве наиболее подходящего решения в рамках существующих ограничений, стоимости разработки и последующего обслуживания.
- Архитекторы (designers) будут разрабатывать проект моста с учетом существующих физических ограничений, налагаемых окружающей средой.

Очень часто заинтересованные стороны выражают проблему уже в терминах предопределенного решения. Это привносит дополнительную работу аналитику, поскольку теперь ему необходимо анализировать является ли предлагаемое (а фактически, - требуемое) решение наилучшим или это всего лишь лишнее ограничение (недоразумение). Например, заказчики, формулируя проблемы, начинают с того, что им необходимы светофоры. Исполнитель же, в свою очередь, задавая вопросы, лишь потом выясняет истинные цели – необходимо увеличение пропускной способности транспортного потока при минимизации риска для водителей и пешеходов, что и является формулировкой проблемы независимой от ее конкретной реализации. После получения такой формулировки проблемы становятся понятными причины выбора последующих решений, которые, возможно, будут подтверждены при помощи моделирования, которое, в свою очередь, будет являться основанием для более детальных спецификаций абстрактного описания решения.

В случае покупки готовой системы необходимо принять решение, что использовать в качестве критерия оценки систем – область проблем (пользовательские требования) или область решения (системные требования). Зачастую решение известно заранее и тогда имеет смысл покупать систему, удовлетворяющую системным требованиям. Однако формулировка проблем в чистом виде дает большие преимущества, даже если основой для выбора приобретаемой системы являются системные требования.

Отсутствие четкого разделения между проблемами и решениями, может привести к следующим негативным последствиям:

- недостаточное понимание существующих проблем;
- невозможность определить границы (масштаб) системы и понять какой функционал должен в нее входить, а какой нет;
- доминирование разработчиков и исполнителей в дискуссиях о системе, поскольку единственное описание, существующее для системы, описывает ее в терминах реализации, а не в формулировках проблем;
- невозможность нахождения наилучшего решения из-за ограничений свободы в выборе решения.

По этим причинам в книге разграничиваются пользовательские и системные требования с точки зрения того, как эти разные требования собираются, как они формулируются, как они моделируются.

1.9 Как читать эту книгу

Книга посвящена процессу разработки требований и призвана помочь системным инженерам, работающим в любой области, и, в частности, инженерам по разработке программного обеспечения, лучше разрабатывать требования.
В Главе 1 обсуждалась важность требований и раскрывалась роль и нюансы процесса разработки требований в процессе создания системы.

В связи с большим количеством информативных связей между главами порядок следования материала был выбран таким, чтобы минимизировать количество ссылок на другие главы. Наилучшим способом чтения этой книги является последовательное прочтение всех глав. Однако для того, чтобы сделать эту книгу более полезной и сократить время на поиски нужной информации для тех читателей, которые заглянули сюда, имея конкретную цель, мы приводим краткое описание последующих глав.

Глава 2 - «Общий процесс разработки требований» - описывает общий подход к разработке требований, применяющийся на всех этапах разработки системы. Хотя этот подход и поможет читателям получить хорошее понимание сущности процесса разработки требований, он, тем не менее, остается достаточно абстрактным. В главах 5 и 6 общий процесс описан более детально, с помощью примеров, применительно к пользовательским и системным требованиям.

Глава 3 - «Системное моделирование для разработки требований» - рассказывает о различных методах системного моделирования и их широком использовании. Эта глава также служит для подготовки читателя к материалам глав 5 и 6, где различные методы моделирования на уровне пользовательских и системных требований рассмотрены на примерах.

Глава 4 - «Написание и анализ требований» - посвящена структуризации документов, связанных с требованиями, и формулировке изложения самих требований. В этой главе рассмотрен подход к «языку», с помощью которого формулируются различные типы требований.

Глава 5 - «Разработка требований в области проблем» - описывает основной процесс анализа проблемной области и сбора пользовательских требований.

Глава 6 - «Разработка требований в области решения» - описывает основной процесс получения требований в области решения: от системных требований до требований к подсистемам и компонентам.

Глава 7 - «Расширенные связи и их анализ» - знакомит с дополнительными методами анализа связей, которые направлены на улучшение процесса работы с требованиями. В главе также обсуждаются метрики, которые можно получить с помощью анализа связей.

Глава 8 - «Аспекты управления разработкой требований» - посвящена вопросам управления проектами в организациях различных типов, в контексте управления требованиями.

На рис. 1.12 изображена структура глав книги и связи между ними.

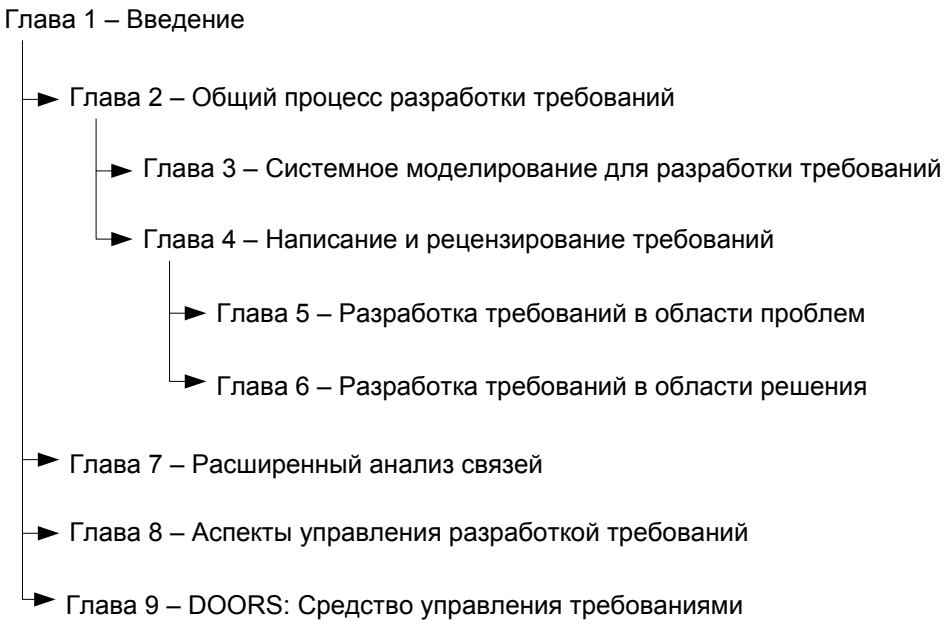


Рис. 1.12 Структура книги

В заключение, Глава 9 - «DOORS: Средство управления требованиями» - описывает DOORS, как конкретный пример программного средства поддержки разработки и управления требованиями.

Процесс, описанный в книге, проиллюстрирован на конкретных примерах, так же как и возможности DOORS.

2 Общий процесс разработки требований

Если вы не можете описать в виде процесса то, что вы делаете, значит, вы не знаете, что вы делаете.
Вильям Эдвардс Деминг (W.E.Deming),
консультант по управлению, 1900–93

2.1 Введение

Эта глава знакомит читателя с концепцией процесса разработки систем.
Она начинается с рассмотрения различных возможных способов разработки. Это помогает выделить общие черты, характерные для многих проектов по разработке, и описать *шаблон разработки систем*, который может использоваться в дальнейшем в различных ситуациях.
В данной главе этот шаблон описывается в виде общего процесса разработки систем лишь с некоторыми подробностями. Последующие разделы показывают то, как общий процесс разработки систем может быть применим в конкретных ситуациях.
Здесь же рассматривается связь между моделями процессов и информационными моделями, а так же выводиться информационная модель для общего процесса разработки.

2.2 Разработка систем

Прежде чем приступать к разработке любой системы необходимо сначала определить - для чего эта система нужна. Если же цель создания системы не определена, то совершенно непонятно какой система должна быть и невозможно даже предположить устроит ли разработанная система ее пользователей.
Форест Гамп прекрасно выразил эту идею, когда сказал:

Если вы не знаете куда идете, то вы вряд ли туда дойдете.

Точность и ясность, с которой формулируются первоначальные потребности, зависит от конкретного человека, который их выражает, и от его позиции в организации.
Изначально потребности могут быть выражены весьма смутно. Например, «мне нужна система, которая бы увеличила эффективность работы моего отдела». Понятно, что такая «характеристика» не является достаточной для того, чтобы выйти с ней на рынок и купить систему. Однако с такой характеристики можно начинать изучение того, что же все-таки человеку нужно. Необходимо исследовать - что в отделе выполняется неэффективно; определить - что именно должно быть улучшено, и выяснить - какими свойствами должна обладать система, чтобы улучшить работу отдела.
Действия, в результате которых из нечеткой первоначальной характеристики потребностей получаются требования, которые могут служить в последствии основой для приобретения или создания системы, и формируют процесс разработки пользовательских требований (stakeholder requirements).

Как уже говорилось выше, к заинтересованным сторонам (stakeholders) могут относиться не только те лица, которые будут непосредственно взаимодействовать с системой, но и те люди и организации, которые так или иначе заинтересованы в существовании системы. Подробно разработка пользовательских требований освещается в Главе 5.

Рис. 2.1 иллюстрирует процесс разработки.

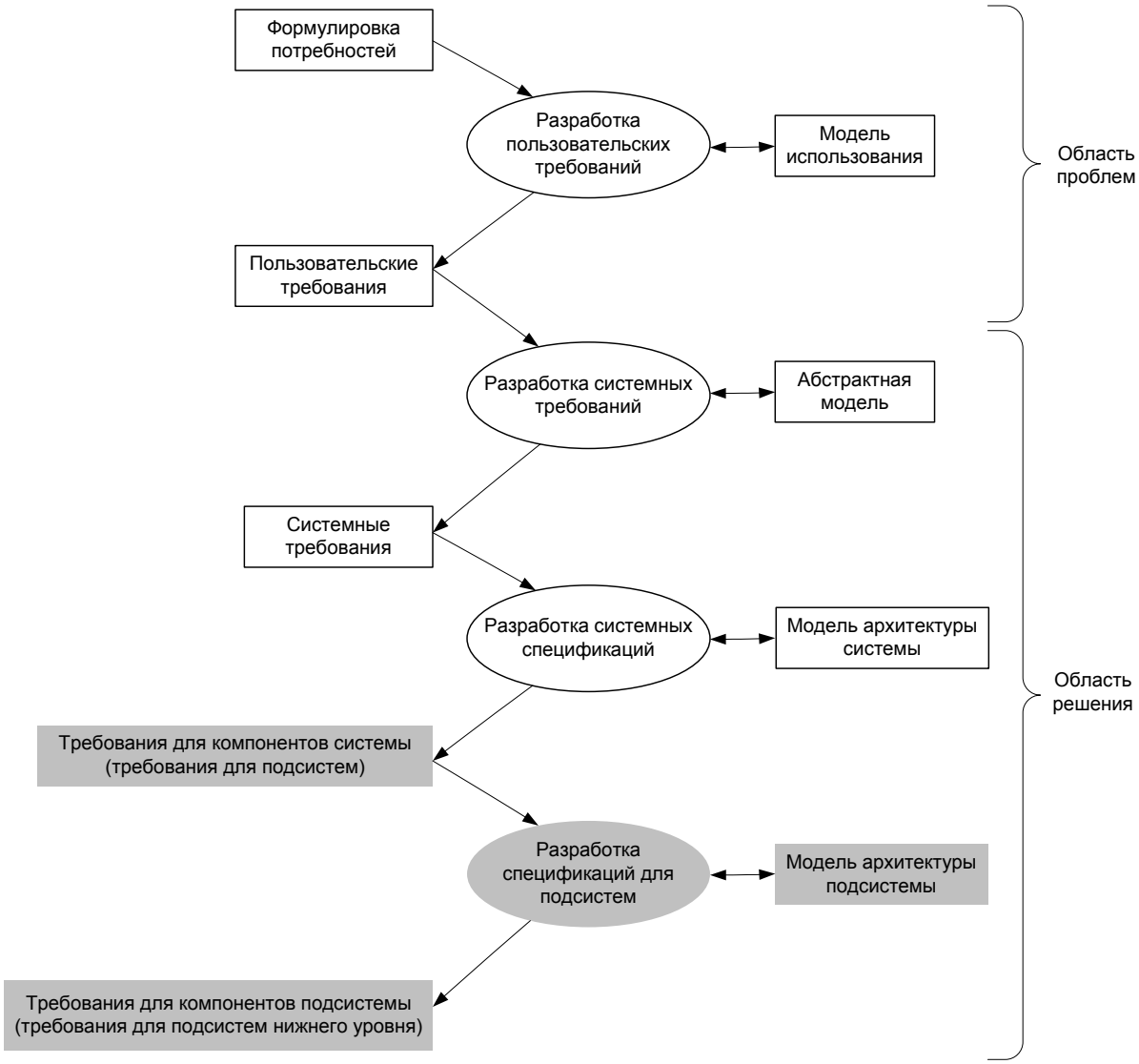


Рис 2.1 Процесс разработки системы.

На приведенной диаграмме для моделей процессов используются следующие графические элементы: с помощью круга (овала) обозначаются процессы, а с помощью прямоугольника обозначаются данные. Стрелки указывают на суть операции с данными - данные читаются или записываются. Так на рис 2.1 показано, что процесс *разработки пользовательских требований* читает (получает) *формулировки потребности* пользователей и записывает (производит) *пользовательские требования*.

Помимо этого процесс *разработки пользовательских требований* производит и читает (использует) *модель использования*.

Думать о потенциальном решении проблемы можно лишь только после того, как разработан полный набор пользовательских требований, поскольку это означает, что уже определено именно то, что пользователь сможет получить от предполагаемой системы.

Далее хорошая практика заключается еще и в том, что вместо того, чтобы сразу переходить к созданию детальных спецификаций системы, нужно сначала определить какие именно характеристики должна иметь система, не вдаваясь в детали реализации.

В этом, как раз, и состоит процесс разработки системных требований. На этом этапе рекомендуется разработать абстрактную модель системы. Несмотря на абстрактность, такая модель очень полезна при обсуждениях внутри команды, разрабатывающей систему, поскольку помогает достичь взаимопонимания и общего (единого) видения будущей системы.

Модель также полезна для объяснения концепций решения тем пользователям и другим заинтересованным сторонам, которые хотят убедиться, что проект движется в правильном направлении. Модель также помогает представить структуру требований в виде документа. Каждый элемент модели может формировать раздел документа. Таким образом, каждое требование попадает в соответствующий контекст, что очень помогает рецензировать готовый набор требований с точки зрения его полноты и завершенности.

Системные требования допускают рассмотрение несколько альтернативных вариантов архитектуры системы. Архитектура определяет набор компонентов, которые, взаимодействуя, порождают требуемые характеристики системы. Эти характеристики являются системными свойствами (emergent properties), которые должны в точности соответствовать требуемым характеристикам системы, описанным в системных требованиях. Модель архитектуры определяет, ***что*** каждый компонент системы должен делать и ***как*** компоненты системы должны взаимодействовать между собой, чтобы система удовлетворяла заданным системным требованиям. Иными словами, архитектурная модель определяет требования для каждого системного компонента (см. рис. 2.1) с точки зрения функций, которые выполняются компонентом, и обязательств его взаимодействия с другими компонентами. Архитектурная модель системы, а, следовательно, и системные требования для компонентов должны также обуславливать и другие свойства системы, такие как физические размеры, производительность, надежность, и удобство эксплуатации (maintainability).

Для большинства систем (за исключением совсем уж маленьких) компоненты архитектурной модели могут быть достаточно сложными для того, чтобы реализовывать их как единое целое. В этом случае компоненты зачастую называют «подсистемами», поскольку их сложность позволяет рассматривать их как самостоятельные системы со своими собственными правами. Но это лишь вопрос терминологии, поскольку даже в этом случае эти подсистемы все равно являются лишь составной частью (компонентом) для системы более высокого уровня.

Процесс разработки архитектурных моделей для каждой подсистемы и последующего получения требований для каждого компонента подсистемы выглядит так же, как и описанный выше процесс разработки архитектурной модели для самой системы. В конечном итоге все сводится к тому, что необходимо разработать архитектурные модели для каждой подсистемы и для всех входящих в них компонентов, как это показано на рис. 2.1.

Описанный процесс разработки также показывает, что разработка системы происходит на разных уровнях, и что на каждом из них выполняются различные действия. Так рис. 2.1 демонстрирует, что процесс разработки на каждом уровне поддерживается моделями (модель использования, абстрактная модель, архитектурные модели), но природа моделей на каждом из уровней существенно отличается. Это пример распространенного подхода – на каждом уровне разработки используется своя модель. Последующие разделы данной главы описывают другие общие идеи, на которых базируется общий процесс разработки требований.

При этом весьма важно понимать, что на каждом уровне существуют собственные требования:

- формулировка потребностей (необходимости);
- пользовательские требования;
- системные требования;
- требования для компонентов системы;
- требования для компонентов подсистем.

Следовательно, разработка требований это не что-то такое, что можно сделать один раз и потом забыть. Работа над требованиями происходит на каждом отдельном уровне и часто происходит на разных уровнях одновременно.
Так, начиная с уровня требований к системным компонентам, происходит множественная и одновременная работа над требованиями на каждом нижнем уровне (на рис. 2.1 это выделено серым фоном).

2.3 Контекст общего процесса

На рис. 2.2. представлен иной взгляд на процесс разработки требований.
Диаграмма демонстрирует, что один и тот же процесс - процесс *разработки требований* - применяется на каждом уровне, хотя ранее были описаны различия в работе на разных уровнях процесса разработки системы. Такой странный, на первый взгляд, подход описания процесса разработки всего лишь говорит о том факте, что существует много общего в работе, выполняемой на разных уровнях.
И хоть целью данной главы является описание таких вот общих аспектов и представление общего подхода к процессу разработки требований, тем не менее, описание процессов будет включать и различия в них, что даст более широкие возможности использования полученных знаний.

Важно отметить тот факт, что при многоуровневой разработке, каждый уровень требует соответствующих знаний и опыта (экспертизы).
На верхнем уровне очень важно знание предметной области, которая соответствует области проблем. На системном уровне важно получить общее представление о системе, чтобы избежать слишком узкого (чересчур детализированного) понимания пользовательских требований. На этом уровне неизбежно появляется уклон в сторону определенного решения. На этой стадии целесообразно и необходимо привлекать людей (организации), которые до этого имели опыт успешной реализации аналогичных систем. Аналогичным образом, разработчики подсистем должны привносить свои знания и опыт, ранее полученные при разработке похожих подсистем.

Следовательно, вряд ли одни и те же люди будут участвовать в разработке на каждом из уровней. Даже если представители одной организации работают на всех уровнях, скорее всего, это будут разные люди, часто из разных отделов, или подразделений.
Поэтому очень полезно придерживаться той идеи, что на любом из уровней работа ведется с единственной целью - удовлетворить условного «заказчика» с более высокого уровня, для чего необходимо привлекать к работе «поставщиков», находящихся уровнем ниже.

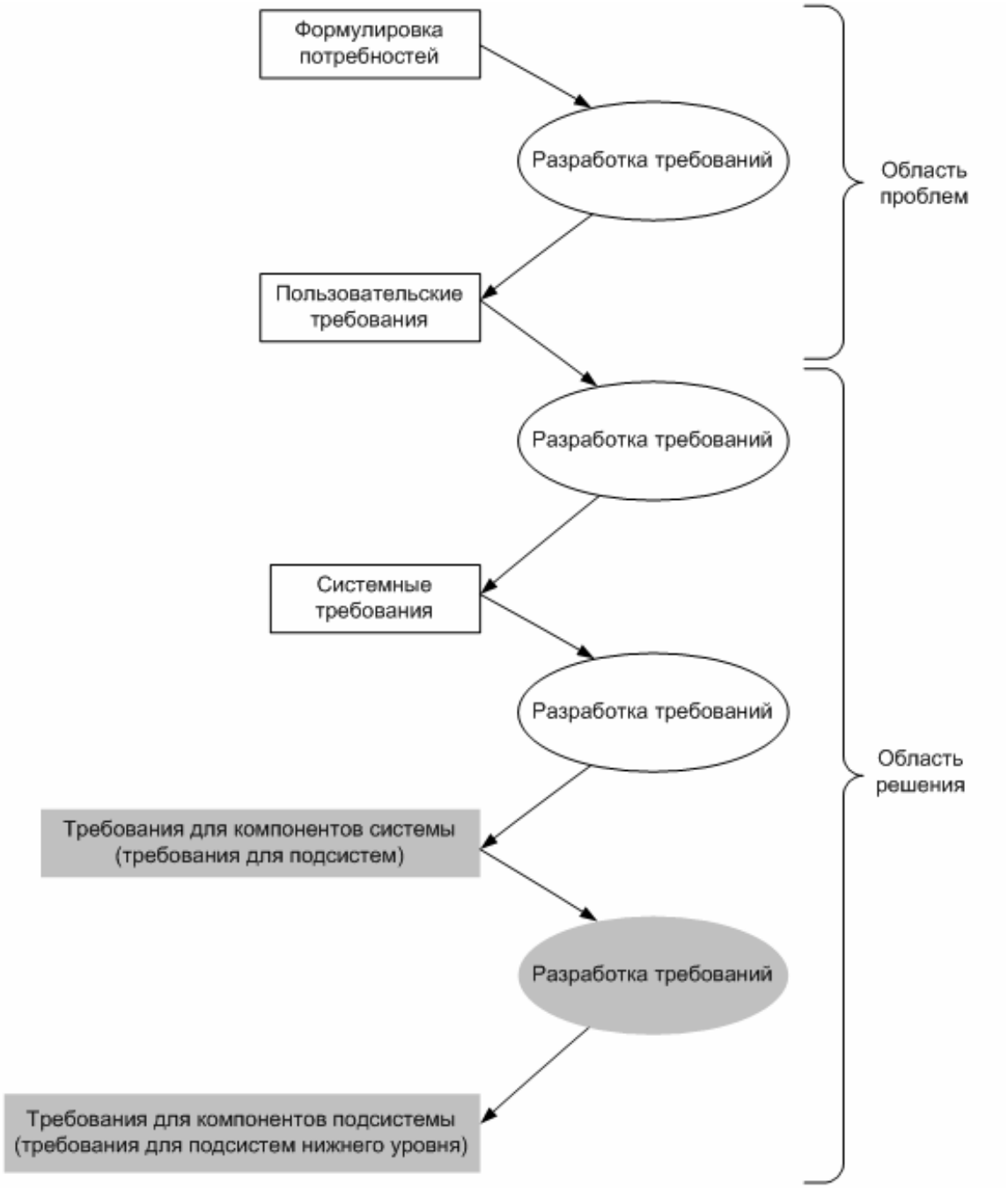


Рис. 2.2 Различные уровни разработки требований.

2.3.1 Входящие и производные требования

Рис. 2.3 является альтернативным представлением рис. 2.2.

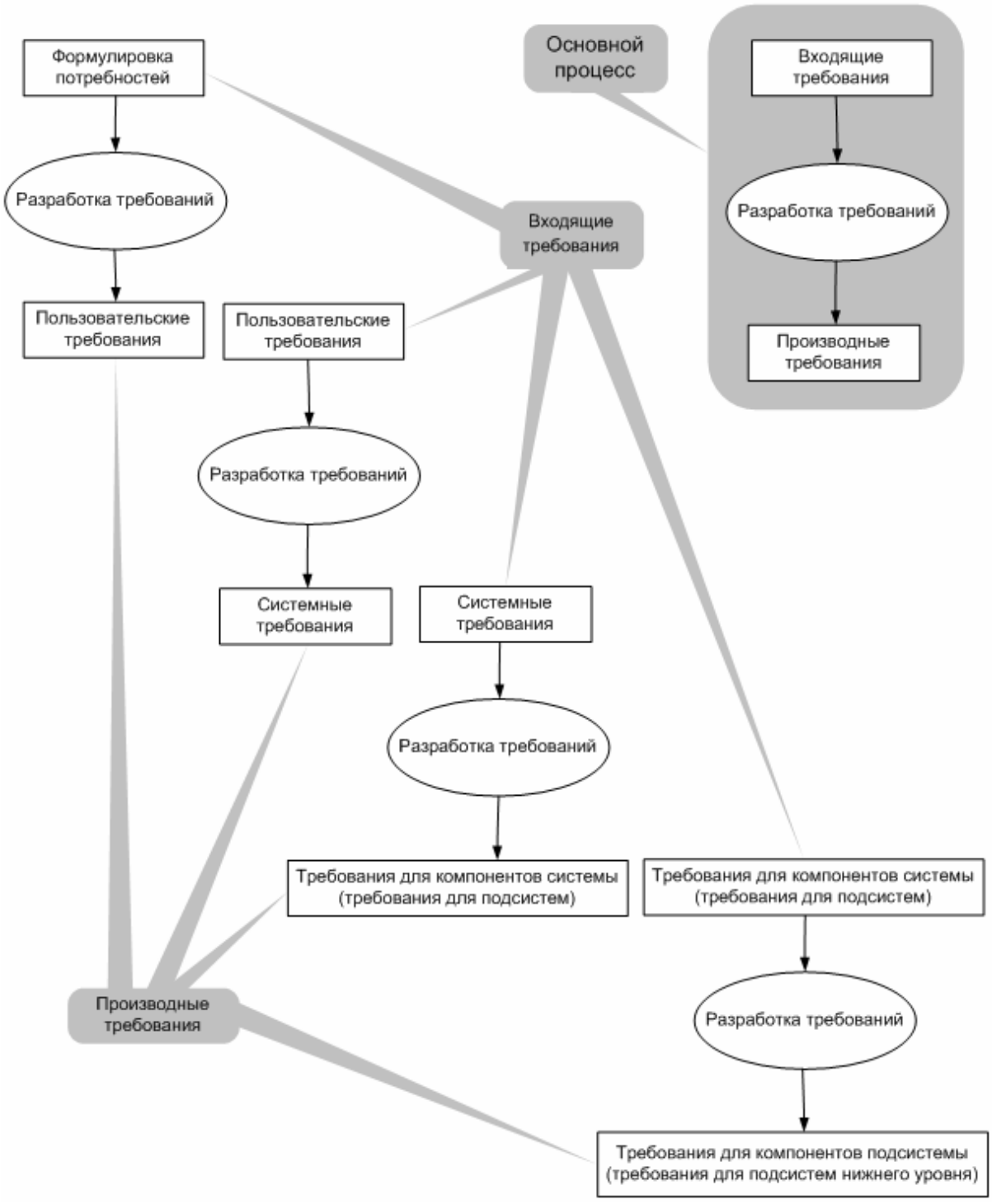


Рис.2.3 Определение входящих и производных требований для основного процесса

На рис. 2.3 каждый индивидуальный процесс детализирован, что подчеркивает тот факт, что требования, получаемые из одного процесса, становятся входящими для другого процесса.
Это естественным образом подводит нас к идее, что процесс разработки требований - это процесс преобразования входящих (с более высокого уровня) требований в производные требования, которые, в свою очередь, являются входящими для следующего уровня.

2.3.2 Стратегия проверки и критерии приемки

Прежде чем объяснять нюансы процесса разработки требований, необходимо обратить внимание на то, что существует другой тип информации, который также является входящим и производным для процесса разработки требований.
Это информация, связанная с проверкой реализации требований.

Для того чтобы полностью осознать всю важность работы с требованиями, и для того, чтобы требования стали хорошей базой для разработки системы, необходимо задуматься над тем, как будущая система (или ее компонент) сможет продемонстрировать, что она удовлетворяет заданным требованиям. Частично это достигается путем формулирования для каждого из требований такого проверочного критерия, который будет впоследствии использован для того, чтобы ответить на простой вопрос - удовлетворяет ли разработанная система требованиям заказчика.

При этом также необходимо определить условия, в которых будет производиться проверка соответствия системы критериям приемки. В первой главе мы знакомились с планами тестирования для каждого уровня разработки требований. Как уже говорилось, тестирование - это только один из типов проверки, но существуют и другие – опытные и стендовые испытания, инспекции и рецензирование. Применяемые типы проверки во многом зависят от характера системы. Например, система, которая связана с угрозой жизни и здоровью, должна проверяться намного более тщательно, нежели та, которая связана с управленческой отчетностью.

Как обобщение вышесказанного на рис. 2.4 показан полный контекст процесса разработки требований.

Часто бывает так, что стратегия создаваемой проверки сама полностью формирует требования для тестового оборудования или добавляет новые требования к уже используемым тестовым установкам, или требует включения в систему дополнительных диагностических функций или контрольных точек. Зачастую это даже преобразуется в отдельный проект для разработки тестового оборудования и других необходимых приспособлений. Например, для авиационного электронного оборудования необходимо выполнить как можно больше тестов до того, как оборудование будет установлено на самолет (по причинам безопасности и себестоимости). И все равно, - даже после установки оборудования на самолет, - необходимо выполнить симуляцию функционирования таких (всех) устройств на земле до начала испытательных полетов. До выполнения первого полета, летчик-испытатель должен быть уверен, что оборудование надежно и работает в соответствии с принятыми нормами и стандартами.

На нижних уровнях, стратегия проверки может определять, должен ли поставщик проверять каждую поставляемую деталь (компонент). Возможные стратегии предусматривают проверку каждого элемента до поставки, проверку групп элементов, или случайную выборочную проверку элементов потребителем.

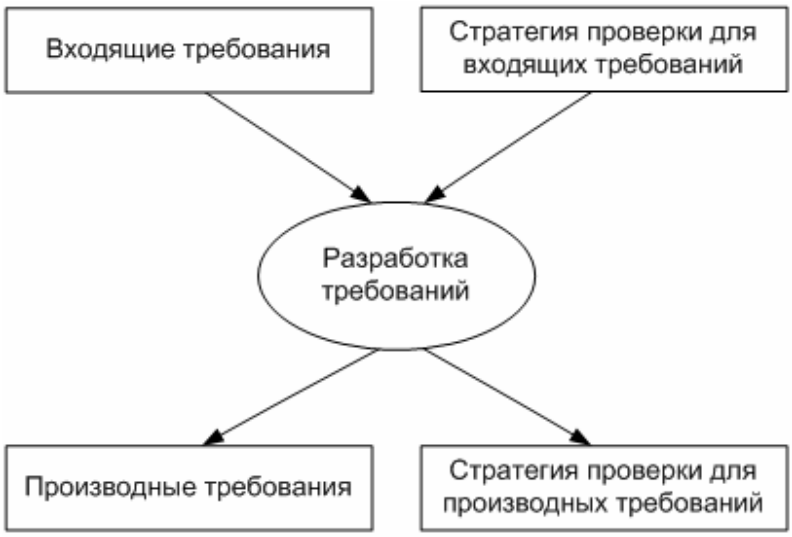


Рис. 2.4 Значение стратегии проверки

2.4 Введение в основной процесс разработки требований

Установив контекст процесса разработки требований, мы может теперь заглянуть и внутрь процесса.
Сначала мы опишем процесс разработки требований для идеального мира, в котором никогда и ничего не меняется, а затем подкорректируем этот процесс, памятуя, что мы живем в постоянно изменяющемся мире.

2.4.1 Идеальный мир

На рис. 2.5 показан процесс разработки требований для идеального мира.
Процесс начинается с необходимости согласования входящей информации (требований) с заказчиками, находящимися на уровень выше.
На втором этапе необходимо провести анализ входящей информации и понять, как из нее получить требуемую выходную (производную) информацию. Этот этап обычно проходит параллельно с первым и подразумевает создание моделей и аналитических отчетов, которые являются основой для разработки производных требований и стратегии проверки требований следующего (нижнего) уровня.
Как только производные требования становятся в достаточной степени сформированными необходимо начать их согласование с исполнителями, работающими на следующем уровне.

На рис. 2.5 показано, что при таком подходе возможна разработка нескольких пакетов производных требований. При этом каждый из этих пакетов должен быть согласован с соответствующим исполнителем, что отнюдь не исключает того, что один исполнитель может отвечать за реализацию несколько пакетов требований.

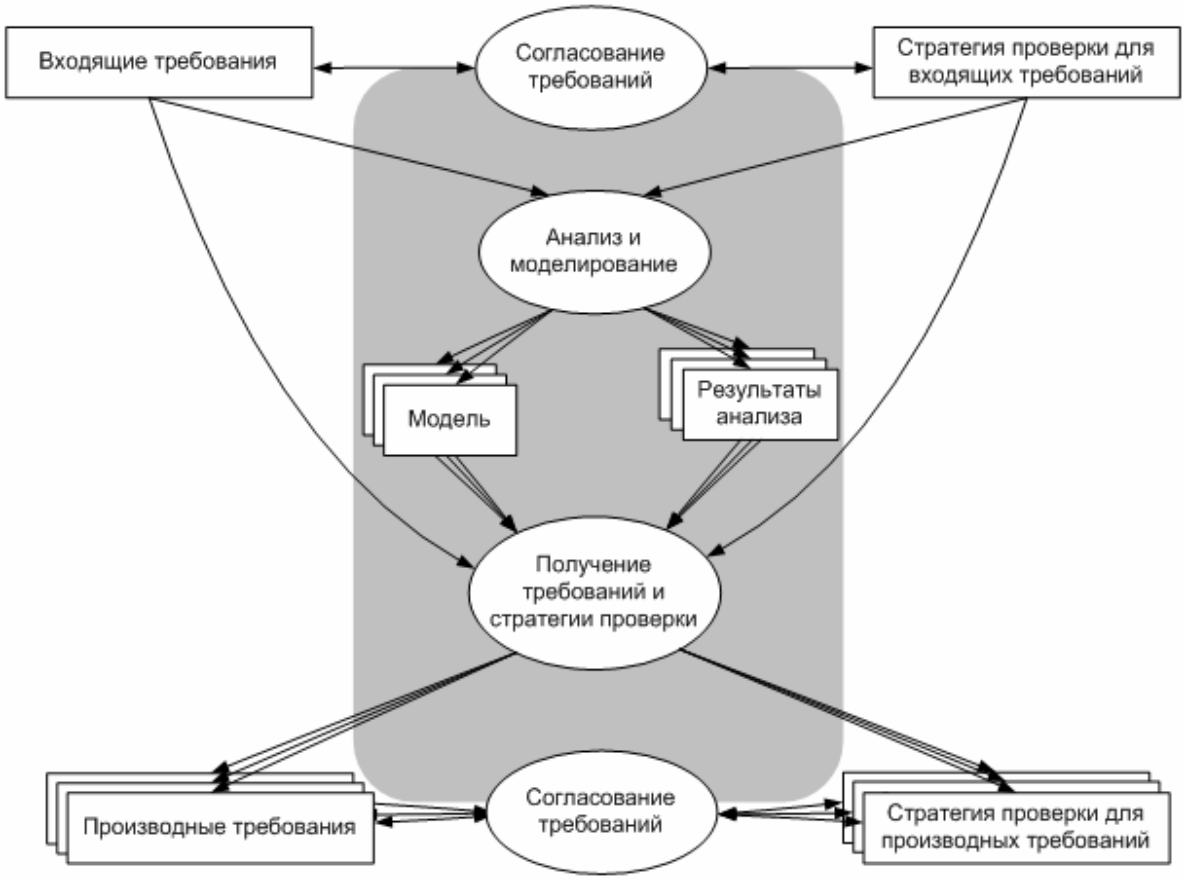


Рис. 2.5 Процесс разработки требований для идеального мира

2.4.2 Разработка требований в контексте изменений

К сожалению или к счастью, но жизнь никогда не стоит на месте. Это особенно сильно заметно в области разработки систем. Создается такое впечатление, что все только и делают, что постоянно меняют свое мнение, утверждая, что-то, что было согласовано ранее, теперь просто категорически неприемлемо.

А раз так, то, следовательно, и процесс разработки требований необходимо адаптировать к изменениям, так как показано на рис. 2.6.

Степень формальности отношения к изменениям зависит от характера проекта и от стадии, на которой он находится.

На ранних стадиях проекта, чтобы проект быстрее двигался вперед, изменения должны предлагаться чаще и приниматься легче.

После того, как требования формально согласованы, обычно рекомендуется уже более строго (формализовано) подходить к принятию и внесению изменений, для того чтобы изменения не вносились по прихоти, например, какого-либо одного из участников проекта. Для этого используется формальная процедура, когда изменения вначале предлагаются (но

не вносятся), а затем рассматриваются в контексте их возможного влияния, которые они могут оказать на весь проект в целом.

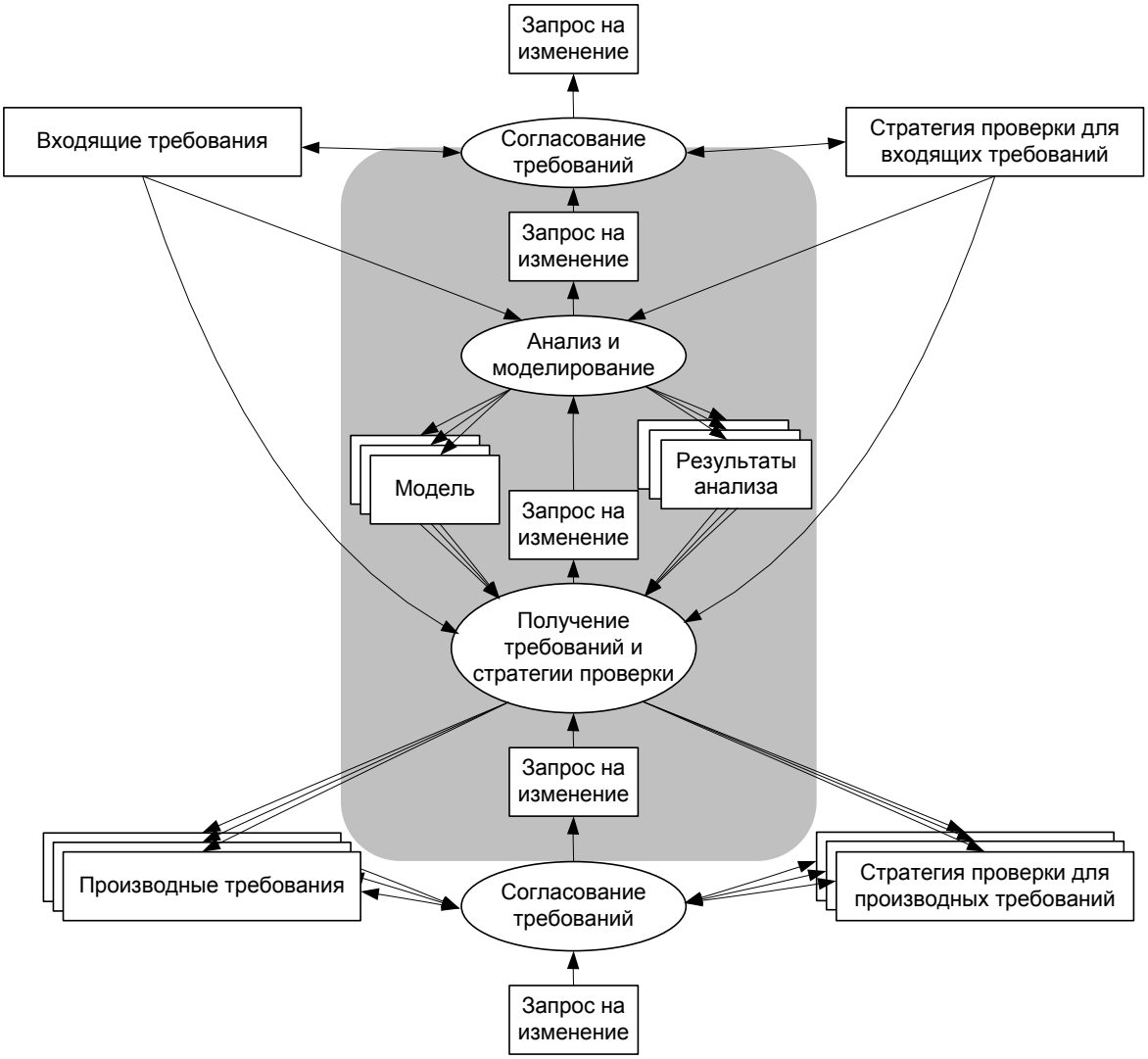


Рис. 2.6 Процесс разработки требований в контексте изменений

Процесс оформления решения по поводу принятия внесения или отклонения конкретного изменения обычно требует участия руководителя проекта и\или группы по контролю изменений. Как уже говорилось, степень формальности, с которой эти люди должны рассматривать конкретное изменение, зависит от характера проекта.
Тема контроля изменений, как часть процесса управления проектом, будет рассмотрена более подробно в Главе 8.

Как следует из рис. 2.6, практически любое действие в рамках процесса разработки требований может привести к изменениям, и эти изменения обычно распространяются вверх. Это не означает, что заказчик никогда ничего не меняет, что все проблемы находятся на нижних уровнях, и «вытекают» из того, что разработка ведется «сверху вниз». Просто

движение сверху вниз уже учтено в рамках основного процесса и теперь необходимо также учесть и обратную связь.
Одна из возможных ситуаций, когда может возникнуть необходимость внесения изменения, это, например, обнаруженные ограничения модели или проблемы, аналитически выявленные при или попытке получения производных требований, или даже при разработке стратегии проверки для производных требований.
Запрос на изменение (или предложение изменения) (change request) может предусматривать не только изменение модели\-ей, но и\или проведение дополнительного анализа для более полного исследования проблемы. Также на этапе *анализа и моделирования* могут быть найдены проблемы, связанные с входящими требованиями, и поэтому создан запрос на внесение изменения в процесс *согласования требований*.

2.5 Информационная модель общего процесса разработки требований

Прежде чем перейти к рассмотрению процессов, протекающих в рамках основного процесса *разработки требований*, было бы полезно рассмотреть общую информационную модель, которая поддерживает этот основной процесс.

Для отображения основного (общего) процесса обычно используются либо диаграммы, содержащие и обозначения процессов, и данные, либо специальная информационная символика. Стрелки на таких диаграммах показывают направленность информации для каждого из процессов - какая информация генерится, а какая используется.

Задачей информационной модели является отображение существующих типов информации и связей, которые могут или должны существовать между ними.

Будет также полезно рассмотреть диаграммы состояний, использующиеся для описания того, как с течением времени изменяется состояние каждого типа информации, что дает наглядное представление о том, когда и как происходит информационное взаимодействие между процессами.

2.5.1 Классы информации

В контексте общего процесса нами уже рассмотрены следующие типы информации:

- входящие требования;
- производные требования;
- стратегия проверки для входящих требований;
- стратегия проверки для производных требований;
- запрос на изменение.

На рис. 2.7 эти пять типов информации представлены в виде диаграммы классов UML-модели.

Название класса всегда указывается в самой верхней (часто она же и единственная) части прямоугольника, обозначающей класс. Средняя часть (если она существует) содержит название атрибутов, которые может иметь класс. В нижней же части (если она

существует) перечисляются названия операций класса (часто также называемых «методами»), которыми может оперировать класс.

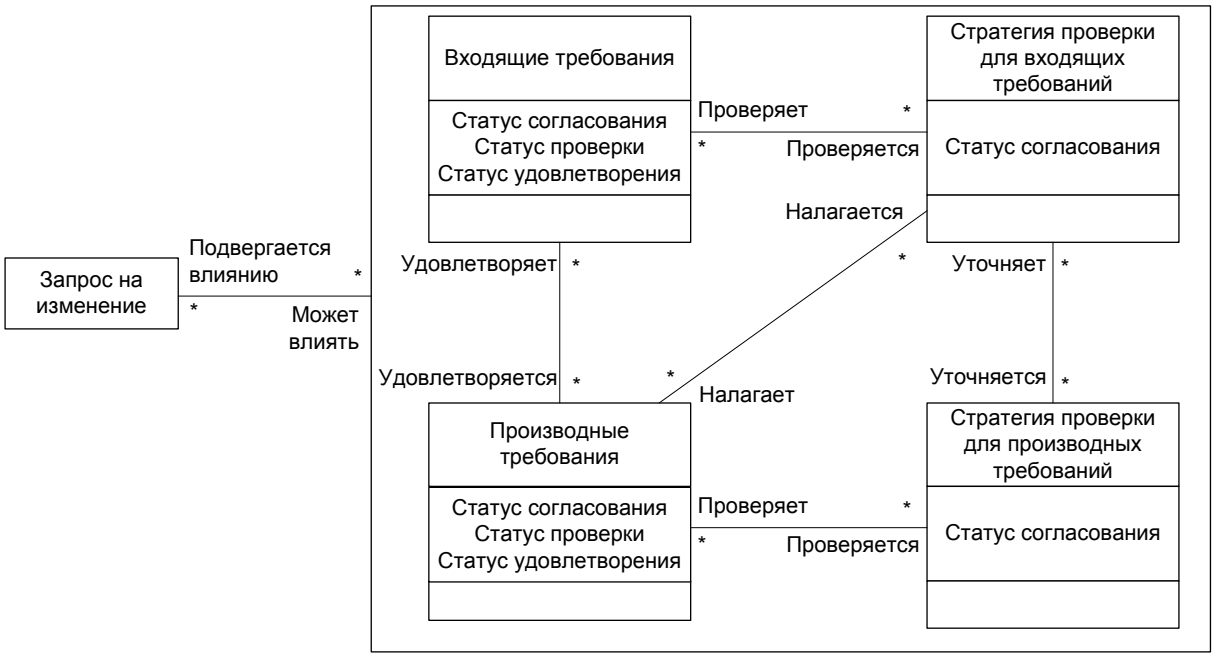


Рис. 2.7 Информационная модель общего процесса

Линии, соединяющие классы, обозначают связи между ними. В UML эти связи также называют «ассоциациями». Как следует из диаграммы, *входящее требование* может быть связано с *производным требованием* связью «удовлетворяется». Аналогично *производное требование* может быть связано с *входящим требованием* связью «удовлетворяет». (Обозначение связи в UML также называют «ролью».)

Звездочкой обозначается, что ноль или более экземпляров класса могут быть вовлечены в ассоциацию. Звездочки на обоих концах ассоциации обозначают, что ассоциация имеет тип многие ко многим.

Следовательно, в модели на рис. 2.7 ноль или более *входящих требований* могут быть удовлетворены одним *производным требованием*, и *входящее требование* может быть удовлетворено нулем или более *производных требований*.

Некоторые читатели могут засомневаться в том, что нижнем уровнем должен быть ноль, - ведь в этом случае нет необходимости иметь ассоциацию вообще? Однако это необходимо и вот почему. Представьте себе, что нижний уровень все-таки был бы равен единице, - тогда это означало бы, что *входящее требование* просто не могло бы существовать до тех пор, пока оно не было бы ассоциировано с хотя бы одним *производным требованием*. Естественно, что такая ситуация неприемлема. *Входящие требования* должны существовать еще до того, как появятся *производные требования*. Следовательно, нулевой уровень - это вполне нормальная ситуация, поскольку периодически в процессе работы могут отсутствовать связи между *входящими* и *производными требованиями* (например, на ранних стадиях разработки, когда связи еще не установлены). Тем не менее, руководитель проекта должен следить за тем, чтобы связи устанавливались как можно раньше. С

помощью связей можно оценить, какой объем работы уже выполнен; определить, что все *производные требования* удовлетворяют *входящие требования*, и, наоборот, что все *вхоящие требования* удовлетворяются *производными требованиями*.

Каждый из классов стратегии проверки (правая часть рисунка) оценивает соответствующий тип требований. При этом стратегия проверки *производных требований* может предусматривать большую детализацию, нежели стратегия проверки *входящих требований*. Такой подход, например, может применяться при разработке систем, связанных с потенциальной опасностью для жизни и здоровья людей. В этом случае необходимо делать более детальные низкоуровневые проверки, совокупность которых помогает удовлетворить критерии проверки более высокого уровня.

Как отмечалось ранее, разработка стратегии проверки может привести к необходимости разработки специальных тестовых стендов или установок. В этом случае появляются связи типа *налагает* между стратегией проверки для *входящих требований* и одним или более *производных требований*. Хорошим примером появления связи такого типа может быть случай, когда (чтобы в дальнейшем проверить компонент) возникает необходимость установить некую контрольную точку (monitor point). Такие контрольные точки часто нужны для проверки производительности системы в реальных рабочих условиях (скорость, время отклика, пропускная способность и т.д.).

Запрос на изменение может относиться к любому из перечисленных четырех классов. На диаграмме это показано с помощью прямоугольника, который охватывает все четыре класса, и наличия связи между этим прямоугольником и запросом на изменение.

Как уже упоминалось выше, средняя часть символа класса служит для отражения имеющихся атрибутов класса. Классы требований имеют три атрибута:

- статус согласования;
- статус проверки;
- статус удовлетворения.

Эти атрибуты описываются в последующих разделах с помощью диаграмм состояний. Статус согласования для классов стратегии проверки подразумевает только два возможных значения:

- *согласованно*
- *несогласованно*.

2.5.2 Статус согласования

Диаграмма состояний для статуса согласования представлена на рис. 2.8. На диаграммах данного типа каждый прямоугольник с закругленными углами отражает состояние одного требования в некий момент времени его истории.

Прямоугольник, помеченный заголовком *Оценка*, называется «супер-состоянием» (super-state) поскольку содержит внутри себя другие состояния. Стрелки, связывающие между собой разные состояния, обозначают переходы, вызывающие изменение состояния.

В самом начале требование возникает в состоянии (статусе) *Предложено*. Затем заказчик - как только посчитает формулировку требования достаточно удовлетворительной - посылает это требование поставщику. Статус согласования требования меняется на супер-состояние *Оценка*. В рамках этого состояния поставщик и заказчик могут проводить

многократные согласования требования до тех пор, пока требование не станет окончательно согласованным, - тогда оно меняет свой статус на *Согласовано*.

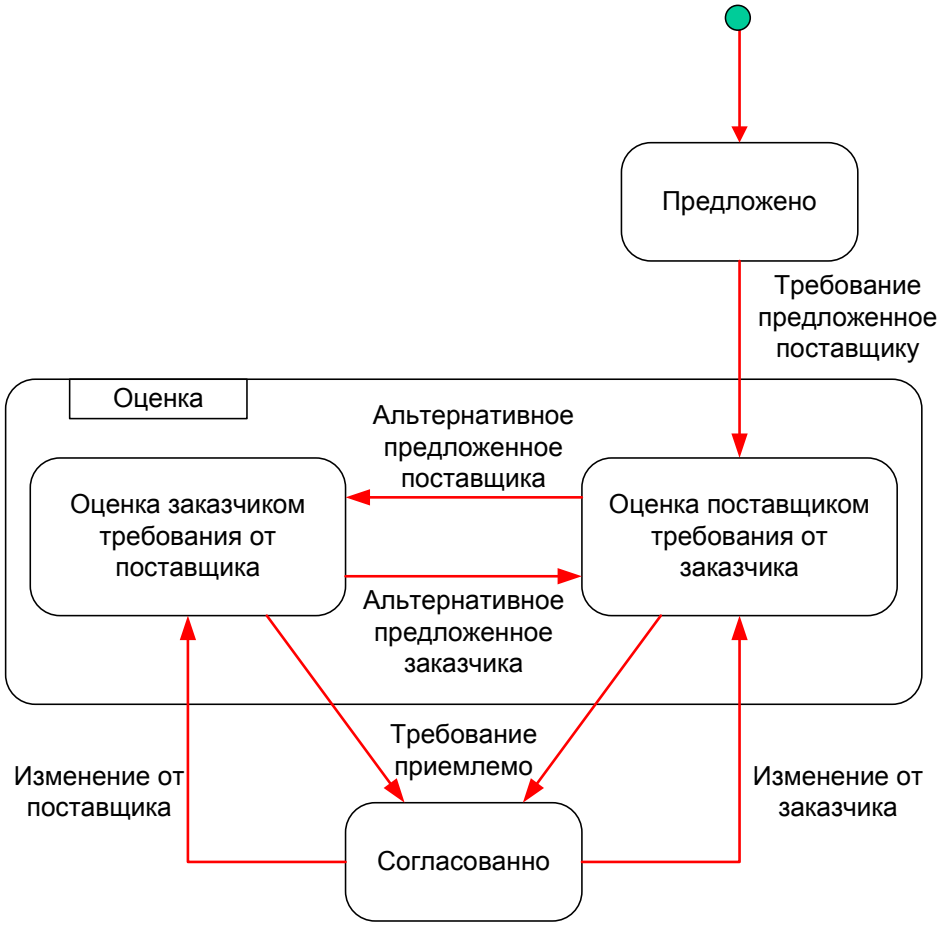


Рис. 2.8 Диаграмма состояний статуса согласования

Согласованное требование остается в этом состоянии до тех пор, пока кто-либо - заказчик или поставщик - не создаст запрос на изменение для этого требования. Если это происходит, то требование вновь возвращается в супер-состояние *Оценка* и имеет этот статус до тех пор, пока вновь не станет согласованным.

Как видно из рисунка, внутри супер-состояния *Оценка* заказчик и поставщик по очереди предлагают альтернативные формулировки требования пока вновь не придут к согласию. Следовательно, внутри супер-состояния *Оценка* требование может находиться **только** в одном из двух состояний - в зависимости от того, какая из сторон в данный момент производит оценку требования.

2.5.3 Статус проверки

На рис. 2.9 представлена диаграмма состояний для статуса проверки требования. Начальное состояние – *Стратегия проверки не определена*.

После согласования стратегии проверки требования, статус проверки переходит в состояние *Стратегия проверки определена*.

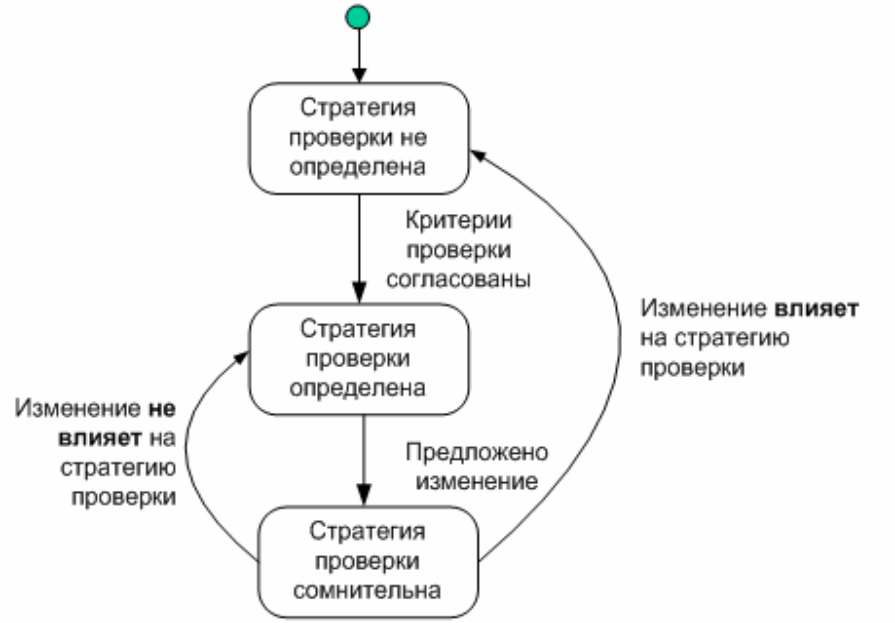


Рис. 2.9 Статус проверки

В этом состоянии статус проверки требования находиться до момента предложения изменения. При этом надо иметь в виду, что изменение может быть предложено как в отношении самого требования, так и для стратегии проверки, связанной с данным требованием.
В случае прихода запроса на изменение, статус проверки переходит в состояние *Стратегия проверки сомнительна* и остается в этом состоянии на время проведения анализа влияния возможного изменения. В результате анализа должно быть четко определено влияет ли предлагаемое изменение на стратегию проверки. Если не влияет, то стратегия проверки остается прежней, а статус проверки возвращается в состояние *Стратегия проверки определена*. В случае, если предлагаемое изменение оказывает влияние на стратегию проверки, то ее необходимо пересмотреть, и в этом случае статус проверки переходит в *Стратегия проверки не определена*.

2.5.4 Статус удовлетворения

Диаграмма состояний для статуса удовлетворения требования представлена на рис. 2.10. Диаграмма очень похожа на диаграмму состояний статуса проверки.

Начальной точкой является состояние *Неудовлетворенно*, что означает отсутствие *производных требований* связанных с данным требованием. После того, как входящее требование удовлетворено одним или несколькими производными требованиями, и при этом «поставщик» с более низкого уровня согласовал требования, а «заказчик» с более высокого уровня согласился с тем, что *производные требования* удовлетворяют *входящее*

требование, статус удовлетворения может смениться на *Удовлетворено*. Необходимо заметить, что возможно потребуется согласовать много *производных требований* для того, чтобы каждое конкретное *входящее требование* имело статус *Удовлетворено*.

Рис. 2.10 Статус удовлетворения

Сразу же после предложения изменения статус удовлетворения переходит в состояние *Удовлетворенность сомнительна*, независимо от того, направлено ли предложенное изменение на требование верхнего или нижнего уровня. Этот статус сохраняется до тех пор, пока не произведен анализ предложенного изменения, и статус удовлетворения не переведен или в состояние *Неудовлетворенно*, или в состояние *Удовлетворенно*.

2.5.5 Внутренние связи информационной модели

Запросы на изменение связывает между собой статусы согласования, проверки и удовлетворения. Получение (регистрация) запроса на изменение меняет сразу же все три состояния (их статус) и приводит к необходимости проведения дополнительной работы: во-первых, - необходимо оценить возможное влияние предлагаемого изменения, а во-вторых, - если влияние существует, - необходимо правильно его адресовать, чтобы принять меры для его адаптации. Необходимо также учитывать, что статус удовлетворения может волнообразно «раскачиваться» вверх-вниз, поскольку все требования связанны между собой связями удовлетворения. Этот «волновой» эффект определяет потенциальные рамки любых побочных изменений, определяя, тем самым, степень «влияние» предложенного изменения.

Статус согласования для *производных требований* должен находиться в соответствии со статусом удовлетворения *входящих требований*, поскольку *входящее требование* не может стать *удовлетворенным* до тех пор, пока поставщик с нижнего уровня не согласует все *производные требования*, удовлетворяющие это *входящее требование*.

2.6 Подробнее об общем процессе

2.6.1 Процесс согласования

Как показано на рис. 2.11, процесс согласования происходит между поставщиком нижнего уровня и заказчиком на уровень выше (см. п. 2.2).

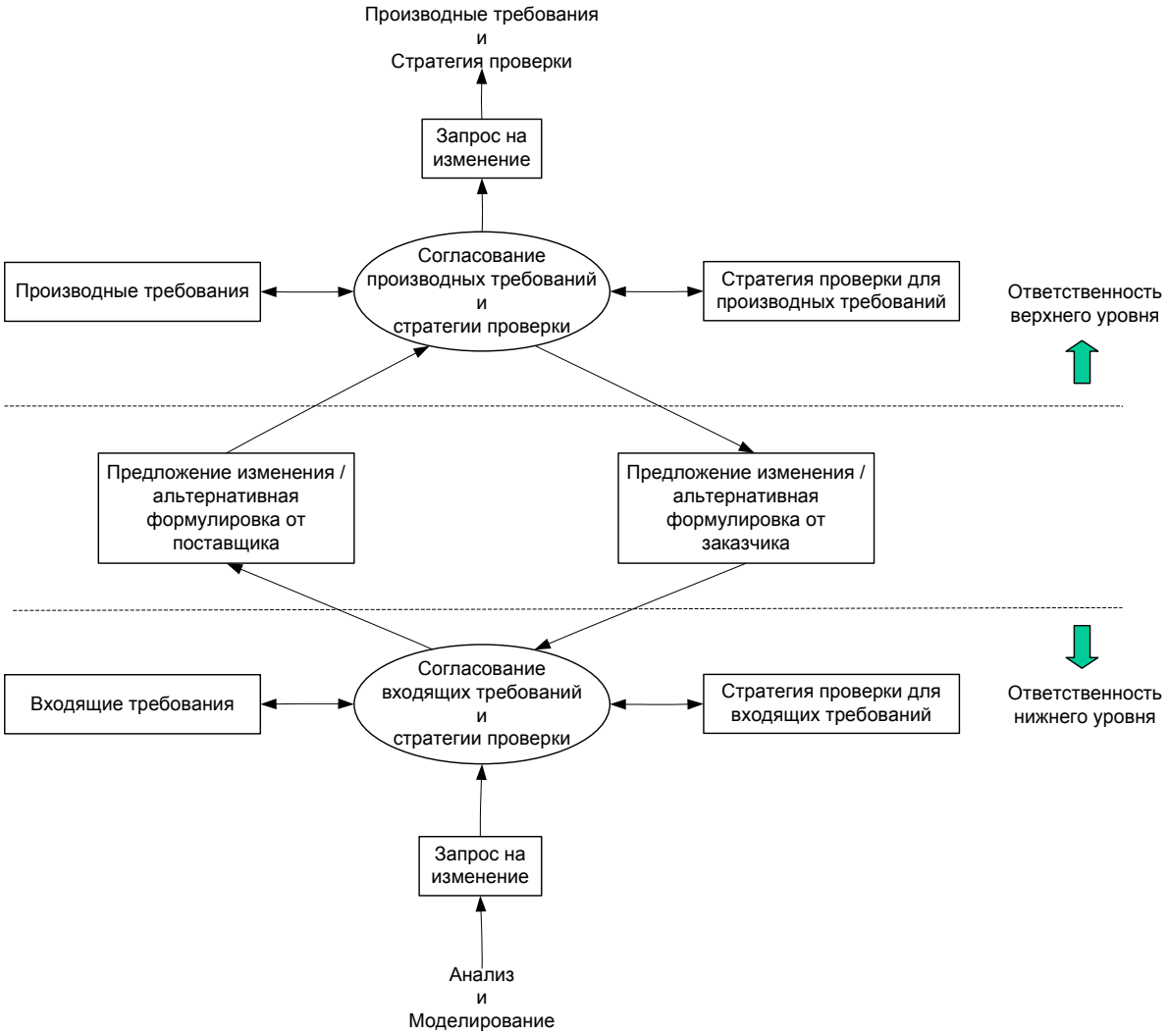


Рис. 2.11 Процесс согласования

Прежде чем приступать к работе с входящими требованиями, необходимо оценить составляют ли они приемлемый «фундамент» для последующей работы.

Чтобы оценить приемлемость требований для работы, необходимо ответить на следующие вопросы:

- Является ли требование полным?
- Является ли требование ясным?

- Является ли требование осуществимым?
- Является ли план проверки требований понятным и приемлемым?

Возможные комментарии к этим вопросам, приведенные ниже, могут служить весомой причиной отклонения любого из требований:

- Отсутствие части информации – зачастую в документах встречаются места, умышленно пропущенные сейчас для заполнения в дальнейшем
  (так в документах, составленных на английском языке, часто встречаются специальные поля с пометкой типа:
  TBA (to be agreed) – необходимо согласовать,
  TBC (to be complete) – необходимо завершить,
  TBD (to be decided) – необходимо принять решение).
- Недостаток ясности – неоднозначность, противоречивость, путаница и т.д.
- Невозможность реализации – никто не знает то, как это сделать.
- План проверки неприемлем.

Если в процессе оценки приемлемости требования, само требование и план его проверки признаются приемлемыми, то статус требования может стать «Согласованно».
Если требование неприемлемо, то поставщик должен направить заказчику альтернативную формулировку требования. В этом случае ответственность («мяч») переходит к заказчику и статус согласования (см. рис. 2.8) переходит в состояние «Оценка заказчиком требования от поставщика». Если заказчик согласен с предлагаемой поставщиком формулировкой, то он может установить статус согласования требования в «Согласованно». В противном случае заказчик сам должен направить поставщику альтернативную формулировку требования. Статус согласования в этом случае переходит в состояние «Оценка поставщиком требования от заказчика», и ответственность («мяч») возвращается к поставщику.

Процесс предложений и контрпредложений продолжается до тех пор, пока не будет достигнуто окончательное соглашение (конечно, теоретически возможна ситуация, когда соглашение не будет никогда достигнуто и споры будут продолжаться бесконечно).

Если в какой-то момент одна из сторон предлагает изменение для уже согласованного требования, то на стороне, получившей запрос на изменение, статус согласования переводится в супер-состояние «Оценка». Далее процесс переговоров продолжается, как описано выше, до тех пор, пока не будет достигнуто новое соглашение.

В течение процесса согласования заказчик может предложить изменения для *производных требований*. Это, в свою очередь, ведет к необходимости оценки влияния изменений на *производные требования* и на *стратегию проверки* производных требований, а также, возможно, к необходимости изменения одного или нескольких *производных требований*. Конечно, может случиться так, что оценить влияние изменения на этом уровне не представится возможным, - в этом случае оценка влияния изменения должна быть перенаправлена выше, на стадию анализа и моделирования. Это необходимо для того, чтобы вовлечь в процесс принятия решения всех необходимых людей, работающих на каждом уровне.
Другими словами, существует необходимость работать над требованиями **параллельно** на нескольких уровнях в одно и тоже время. Т.е. общий процесс должен выглядеть как последовательность параллельных подпроцессов согласований и подпроцессов принятия

решений. Это полностью разрушает парадигму «водопадного» процесса разработки, в котором все действия происходят в одном, строго заданном, порядке – сверху вниз.

И еще один нюанс. Часто бывает так, что команда, работающая над проектом, начинает задумываться о критериях приемки системы или планах проверки требований достаточно поздно (например, или только после того как все требования утверждены или же согласование приемочных испытаний заканчивается непосредственно перед началом тестирования и т.д.). Это плохая практика. Обычно она ведет к задержкам (и даже срывам) сроков окончания проекта из-за того, что уж согласованные требования приходиться менять на поздних этапах проекта для того, чтобы сделать их, наконец, пригодными для тестирования!

2.6.2 Анализ и моделирование

Аналитическая часть процесса связана, в основном, с пониманием сущности и границ (масштабов) входящих требований с целью последующей оценки возможных рисков, связанных с их удовлетворением (реализацией). Аналитическая работа может варьироваться от простых исследований реализуемости возможных вариантов решения до разработки прототипов наиболее важных частей системы или тех частей, реализация которых связана с наибольшим риском. Так, например, очень часто приходится разрабатывать модель производительности для того, чтобы оценить возможную пропускную способность системы и время отклика.

Моделирование – как часть общего процесса - также используется для понимания сущности и определения структуры производных требований. В большинстве случаев для понимания и структурирования пользовательских требований (stakeholder requirements) используются прецеденты (use cases) или пользовательские сценарии (users scenarios), которые помогают понять то, как люди будут использовать разрабатываемую систему.

Архитектурные модели обычно применяются для определения структуры требований в области решений. Такие модели описывают не только элементы решения, но и то, как они взаимодействуют между собой.

Во многих случаях модели используются для создания архитектуры предлагаемого решения. Такие модели зачастую достаточно однозначны в тех областях, в которых архитектура де-факто имеет место быть (например, автомобилестроение, телекоммуникации, авиастроение). Однако в инновационных областях, где еще отсутствует понятие архитектуры системы как таковой, модель может носить более абстрактный характер и допускать потенциальные альтернативы (варианты реализации).

В основном, использование тех или иных моделей целиком зависит от характера конкретного проекта. Как ранее упоминалось, типы моделей очень сильно зависят от предметной области. Так, например, в области разработки программного обеспечения достаточно широко используются объектные модели. А для сравнения в таблице 2.1 представлены различные типы моделей, которые применяются в трех различных областях промышленности.

Основная цель создания разного рода моделей – это попытка понять суть *входящих требований* вместе с предлагаемой стратегией их проверки, а также последующие эксперименты с альтернативными вариантами реализации этих требований, которые (эксперименты) должны опережать разработку *производных требований*. Эта работа может рассматриваться еще и как возможная разработка стратегий проверки для *производных*

требований, что, в свою очередь, может вести к появлению соответствующих требований для создания тестового оборудования и/или программного обеспечения. Причем эта работа может также служить и источником проверочных требований для *производных требований*.

Таблица 2.1 Примеры методов моделирования

Авиастроение
Аэродинамические модели
Трехмерные пространственные модели
Модели распределения нагрузки
Симуляторы полетов
Железнодорожный транспорт
Симуляторы расписаний
Модели безопасности, надежности и ремонтопригодности
Автомобилестроение
Модели дизайна
Модели приборной панели
Аэродинамические модели

Процесс *анализа и моделирования* (рис. 2.12) может протекать одновременно с процессом *согласования*, поскольку такой подход помогает получить более глубокое понимание требований.

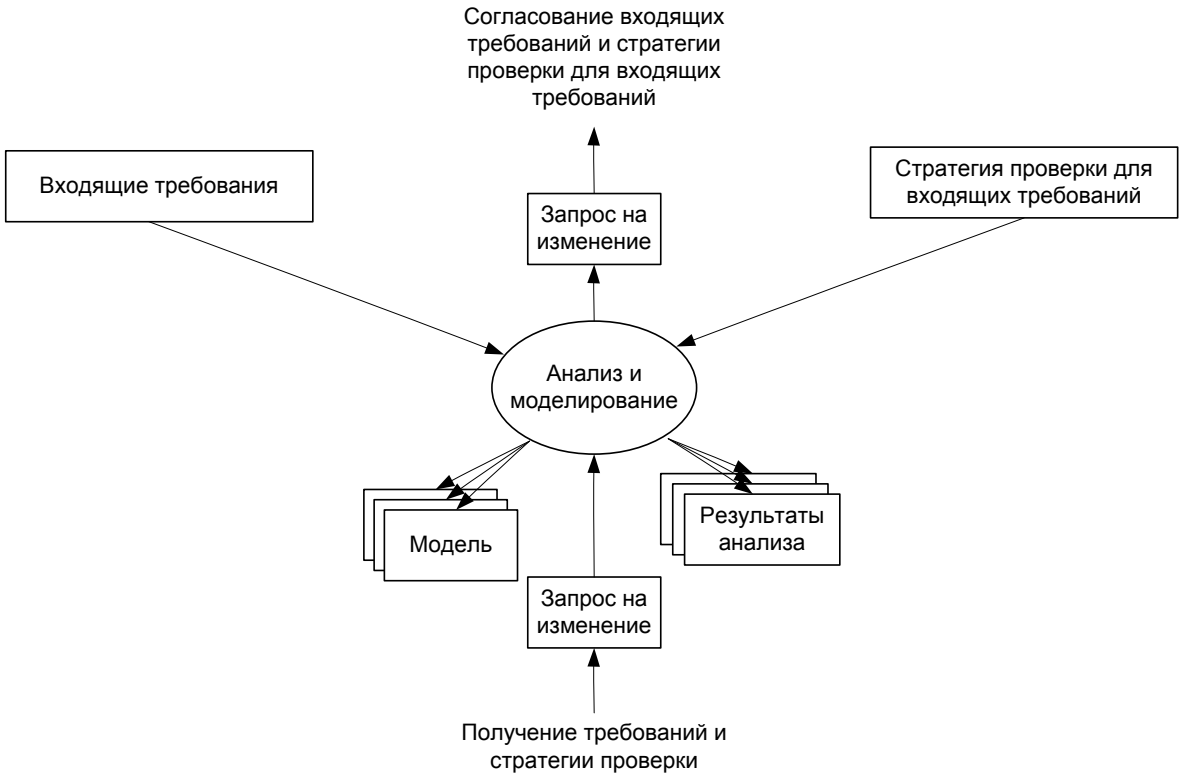


Рис. 2.12 Процесс анализа и моделирования.

Помимо этого, в процессе *анализа и моделирования* очень часто возникает множество вопросов, связанных с формулировкой и пониманием (интерпретацией) *входящих требований*. Это, в свою очередь, ведет к появлению запросов на изменение, которые могут вновь возвращать участников проекта к процессу *согласования*.

В Главе 3 рассматриваются некоторые распространенные методики моделирования, особенно те из них, которые применяются при разработке программного обеспечения.

В Главе 5 объясняется, как использовать модели прецедентов и пользовательские сценарии для улучшения понимания пользовательских требований.

В Главе 6 рассматриваются функциональные модели, которые помогают создавать базу для разработки системных требований.

2.6.3 Получение требований и стратегии проверки

Иллюстрацией этого процесса служит рис. 2.13.

Получение требований

Существует множество различных вариантов использования моделей в целях получения требований. Однако простейший случай, который можно рассматривать как базовый, - это получение требований к компонентам на основании архитектуры системы. В этом случае появляется возможность определить, какие именно требования должны быть удовлетворены каждым из компонентов системы. Некоторые из требований для компонентов системы могут совпадать с одним или более *входящими требованиями*; другие же могут быть получены из входящих требований таким образом, чтобы распределить выполнение *входящих требований* между несколькими компонентами системы. Другой набор требований может быть порождением ограничений, накладываемых архитектурой системы или самими *входящими требованиями*. Этими ограничениями могут быть ограничения конкретных интерфейсов, возможные физические ограничения, такие как масса, объем, потребляемая электрическая мощность, теплоотдача и т.д.

Разумеется, на практике некоторая работа по получению или классификации требований для компонентов системы может выполняться еще и до окончательного согласования *входящих требований* и принятия стратегии их проверки. Однако эта работа ***не может считаться завершенной*** до получения окончательно согласованных входящих требований и стратегии проверки для них.

Следует отметить, что при получении требований для компонентов системы необходимо выполнить определенную дополнительную работу - установить связи типа «удовлетворяет/удовлетворяется» между *входящими* и *производными требованиями*. Такие связи показывают: какие *входящие требования* удовлетворяются какими *производными требованиями*, что может быть использовано для утверждения того, что:

- все *входящие требования* удовлетворены;
- все *производные требования* необходимы (т.е. они прямо или косвенно удовлетворяют одно или более *входящих требований*).

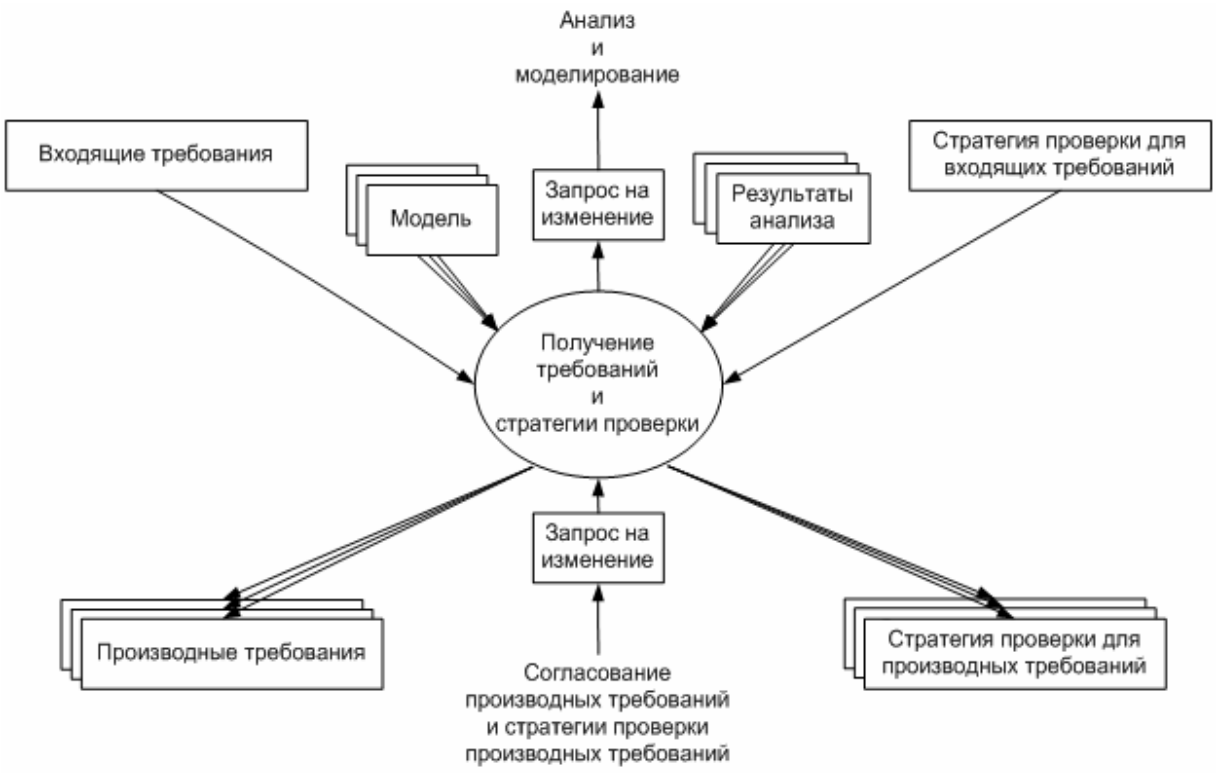


Рис. 2.13 Процесс получения требований и стратегии проверки

При этом абсолютно недостаточно убедиться лишь в самом факте наличия связей типа «удовлетворяет/удовлетворяется», например, при помощи матрицы связей.
Здесь важно, чтобы для каждой из этих связей заранее устанавливалось собственное обоснование – некий мотивирующий аргумент.

Очевидно, что в процессе получения требований из моделей, в последних могут обнаружиться либо пробелы, либо ошибки. Такие случаи ведут к генерации запросов на изменение, поступающих (возвращающихся) к командам, разработавшим ту или иную модель. Соответствующая команда будет вынуждена либо скорректировать непосредственно модель, либо запросить уточняющие разъяснения, либо направить запрос на изменение *входящего требования*. Таким образом, процесс эскалации запроса на изменение может продолжаться.

Получение стратегии проверки

Как обсуждалось выше, связи типа «удовлетворяет/удовлетворяется» отражают процесс получения *производных требований* из *входящих требований* – спецификацию системы. В отличие от этого, стратегия проверки определяет план того, как каждое требование будет тестироваться на каждом уровне.

Стратегия проверки состоит из набора проверочных мероприятий, каждое из которых может быть определенного вида исследованием, тестом или инспекцией. Возможно даже, что несколько проверочных мероприятий будут разработаны для проверки лишь одного требования.

Каждое проверочное мероприятие должно подразумевать следующие аспекты:

- ***тип*** мероприятия, который должен соответствовать требованию;
- ***фаза***, на которой должно проводиться мероприятие – чем раньше, тем лучше;
- специальное ***оборудование***, которое может понадобиться для проведения мероприятия;
- описание того, что будет являться успешным ***результатом*** мероприятия.

План проверки может быть структурирован исходя либо из фазы (стадии проекта), либо из типа проверочных мероприятий.

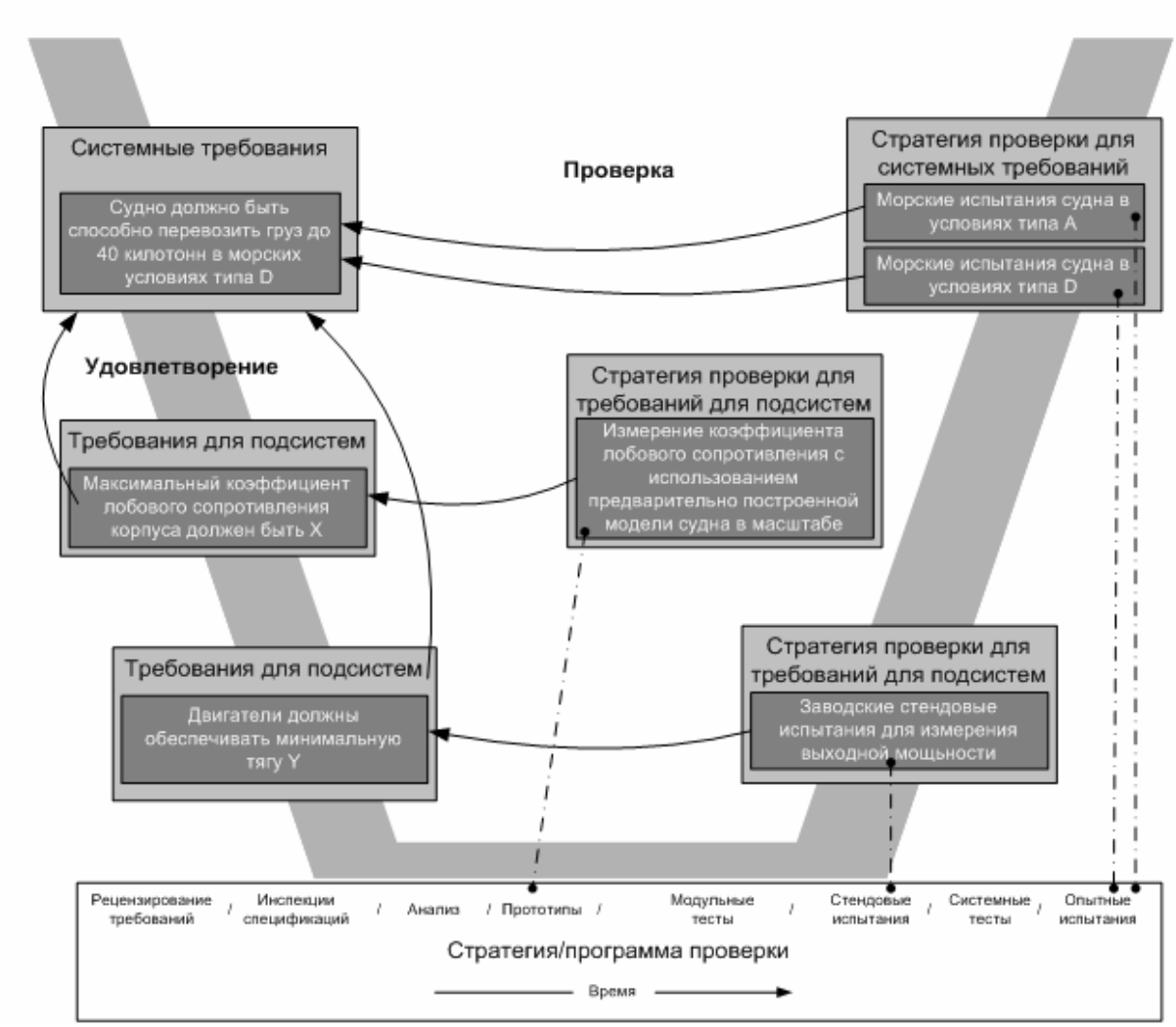


Рис. 2.14 Проверка требований

Проверочные мероприятия должны в обязательном порядке соответствовать конкретному уровню требований. Другими словами, пользовательские требования (требования заказчика) должны служить основой для разработки программы приемочных исследований и испытаний. В то время как системные требования должны служить основой для разработки системных тестов, которые должны предварять опытные испытания. Таким

образом, нет необходимости разрабатывать системные тесты для пользовательских требований, поскольку для пользовательских требований должны существовать полученные из них системные требования, для которых и существуют системные тесты.

Рассмотрим пример, приведенный на рис. 2.14.

Системные требования для корабля разделены на два требования для разных подсистем – для корпуса и силовой установки. Соответственно, два различных теста запланированы для требования на системном уровне, и два других - на уровне подсистем.

Следовательно, для полного понимания того, как требования будут проверяться, необходимо наличие связей «удовлетворяющего» типа и стратегии проверки.
Нетрудно заметить, что для того, чтобы отобразить статус проверки требований высокого уровня, необходимо учитывать результаты проверки связанных требований со всех нижних уровней с помощью связей «удовлетворяющего» типа и связей «проверяющего» типа.

2.7 Заключение

В этой главе мы рассмотрели общий процесс разработки требований, который одновременно применяется на каждом уровне разработки системы. Основным преимуществом общего процесса является то, что он определяет общие действия, характерные для каждого уровня:

- согласование *входящих требований* с заказчиком;
- анализ *входящих требований* для определения рисков и потенциальных проблем, связанных с удовлетворением требований;
- создание одной или нескольких моделей для исследования возможных стратегий получения последующих требований;
- формирование требований, полученных из *входящих требований* при помощи результатов *анализа и моделирования*;
- согласование *производных требований* с командой (командами) сотрудников, ответственных за их реализацию;
- установление связей «удовлетворяющего» типа между *входящими* и *производными требованиями*;
- установление связей «проверяющего» типа между *производными требованиями* и соответствующей стратегией проверки.

Все эти действия приводят к организации информации таким образом, как она была описана в информационной модели общего процесса. Текущее состояние информации используется для измерения прогресса в работе, оценки влияния предлагаемых изменений и определения показателей выполнения проекта.
Например, состояние требования может характеризоваться его тремя атрибутами (свойствами):

- согласование;
- удовлетворение;
- проверяемость.

Идеальное состояние любого требования в любой системе заключается в том, что это требование:

- согласовано между заказчиком и исполнителем;
- имеет согласованную стратегию его проверки;
- удовлетворяется требованиями более низкого уровня (или спецификацией системы).

Степень отклонения состояния каждого из требований от этого идеального статуса характеризует величину риска, которому подвергается проект с точки зрения управления требованиями, и, в тоже время, отражает объем работы, который необходимо выполнить для того, чтобы привести все требования к идеальному состоянию.

3 Системное моделирование для разработки требований

Метод - это перекресток, на котором встречаются искусство и наука.

Эдвард Булвер-Литтон (Bulwer-Lytton),
поэт, 1803–73

3.1 Введение

Системное моделирование вносит дополнительную формальность в процессы анализа и проектирования. В процессе разработки системы очень часто используются схемы и рисунки, которые помогают наглядно отобразить некоторые аспекты разработки. Системное моделирование формализует это наглядное представление не только с помощью диаграмм, выполненных с использованием стандартных нотаций (синтаксиса), но и обеспечивает среду (средства) для понимания и обсуждения идей, связанных с процессом разработки.

Можно смело утверждать, что искусство моделирования является самой творческой частью работы системного инженера. На практике не существует единственного «правильного» решения, поэтому модели меняются и развиваются на протяжении этапов разработки. В большинстве случаев, модели представляют собой некий визуальный ряд, в котором для отображения информации используются взаимосвязанные диаграммы. Новые методы, - такие как, например, объектно-ориентированные методы моделирования, - разумеется, расширяют концепцию моделирования, однако большинство используемых в них подходов, тем не менее, базируются на известных и проверенных временем принципах.

Хорошая модель - это модель, помогающая общению. Модели нужны, для того чтобы можно было обсуждать идеи и внутри команды разработчиков, и во всей организации в целом, подключая к этому процессу и другие заинтересованные стороны. Модели могут применяться для различных целей и покрывать самые разнообразные аспекты разработки системы. Так, например, одна модель может описывать общую структуру взаимодействия внутри всей организации, а другая – отображать всего лишь одно конкретное функциональное требование к этой системе.

Моделирование имеет следующие преимущества:

- Поощряет использование *точно определенной терминологии*, однозначность которой поддерживается в рамках разработки всей системы.
- Позволяет *с помощью диаграмм* получить *наглядное* представление системных спецификаций и архитектуры системы.
- Позволяет рассматривать *различные аспекты взаимодействия* системы с различных точек зрения.
- Поддерживает *системный анализ*.

- Позволяет *подтвердить достоверность* некоторых аспектов поведения системы с помощью динамических моделей.
- Позволяет *постоянно совершенствовать* систему посредством уточнения архитектуры, поддерживая *генерацию тестов и исходного кода*.
- Позволяет свободно *общаться различным организациям* между собой, используя стандартные нотации.

Моделирование позволяет системному инженеру в большей мере проявить свои творческие способности. Данная глава рассказывает о системном моделировании и описывает некоторые методы разработки требований, в которых используется моделирование.

3.2 Методы моделирования для разработки требований

3.2.1 Диаграммы потоков данных

Диаграммы потоков данных (Data flow diagrams или DFDs) являются базовым понятием для многих традиционных методов моделирования. Диаграммы потоков данных имеют достаточно ограниченный набор графических элементов для представления структуры системы и ее интерфейсов. И, несмотря на то, что изначально диаграммы потоков данных предназначались для описания информации и информационных потоков, диаграммы этого типа могут использоваться для описания потоков любого типа, независимо от того базируется система на использовании компьютерной техники или нет. К сожалению, диаграмма потоков данных не отражает один важный поток информации – поток управления.

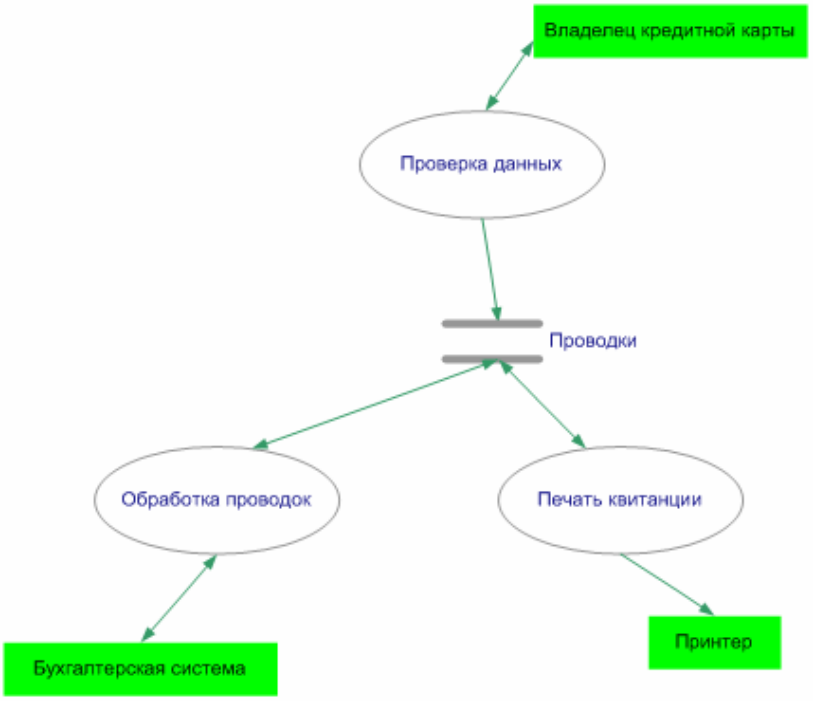


Рис. 3.1 Диаграмма потоков данных.

На диаграммах потоков данных используются следующие элементы:

- потоки данных (обозначаются стрелками с названиями);
- элементы преобразования данных (обозначаются окружностями или овалами);
- хранилища данных (отрезки горизонтальных параллельных линий);
- внешние сущности (прямоугольники).

На рис. 3.1 приведен простой пример традиционного использования диаграммы потоков данных для моделирования информационный системы.

Потоки отображают либо информационный, либо материальный обмен между двумя элементами преобразования. В реальной жизни эти потоки могут быть непрерывными, запускаться по запросу, быть асинхронными и т.д. В соответствии с нотацией диаграмма должна сопровождаться текстовым описанием каждого процесса, хранилища данных и потока данных.

Для определения всех потоков и хранилищ данных используется *словарь данных*. Каждых овал (окружность) определяет базовую функциональность, которую обеспечивает компонент системы. Эта функциональность описывается с помощью *П-спецификаций (P-spec)* или *мини-спецификаций (mini-spec)*, которые представляют собой текстовое описание, зачастую написанное с помощью псевдокода.

Контекстная диаграмма - это диаграмма самого верхнего уровня DFD модели, которая показывает все внешние системы, взаимодействующие с разрабатываемой системой, как это показано на рис. 3.2.

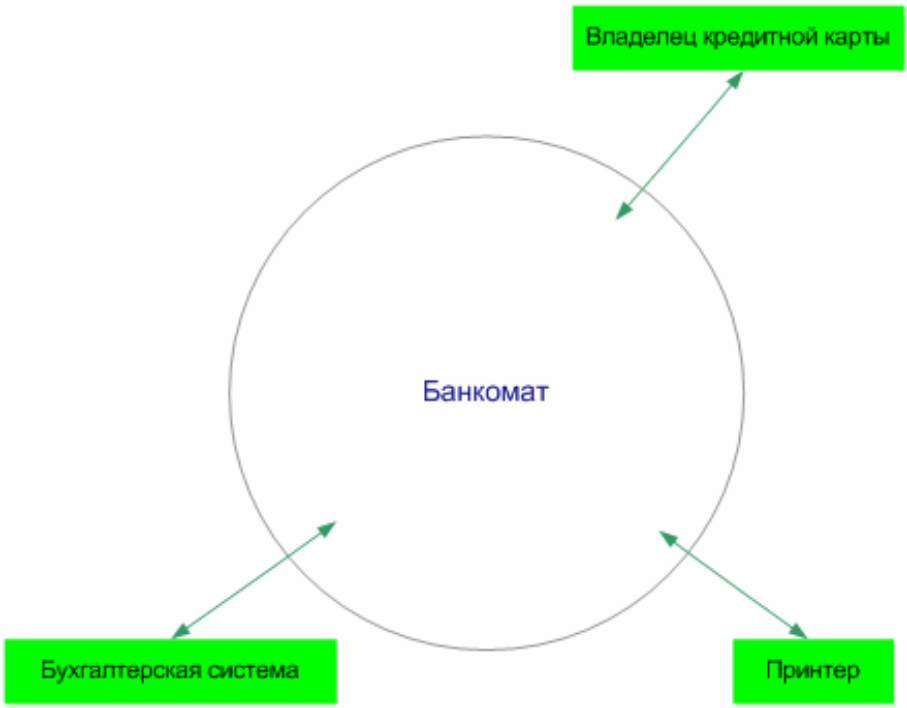


Рис. 3.2 Контекстная диаграмма.

Овалы могут детализироваться при «движении» вниз от уровня к уровню (декомпозиция). При этом функциональное содержимое каждого овала может раскрываться с помощью отдельной диаграммы, которая, в свою очередь, также может содержать другие овалы и хранилища данных (см. рис. 3.3).

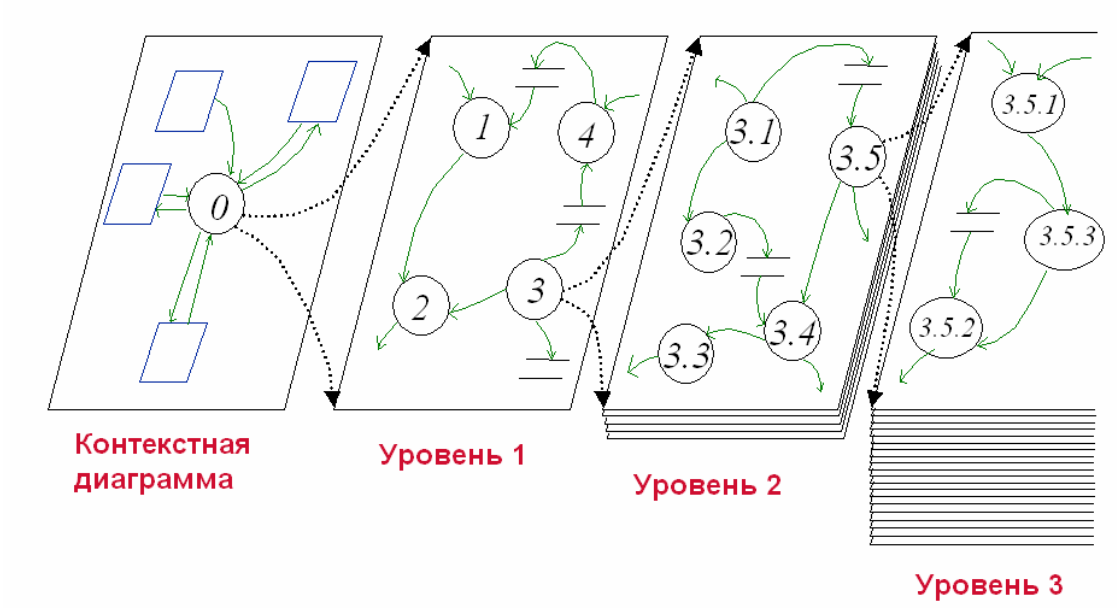


Рис. 3.3 Функциональная декомпозиция.

Для иллюстрации применения диаграмм потоков данных рассмотрим пример контекстной диаграммы для системы управления и контроля скорой помощи (рис. 3.4). Эта диаграмма является отправной точкой для анализа потоков данных системы.

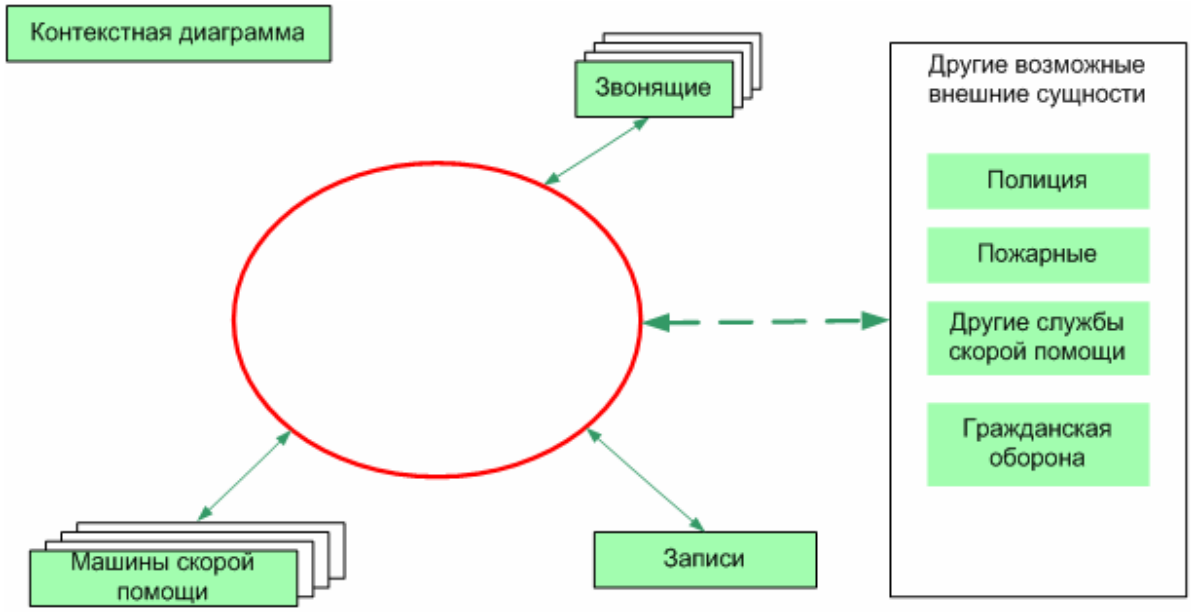


Рис. 3.4 Контекстная диаграмма для системы управления и контроля скорой помощи.

Самые главные внешние сущности для системы это *звонящие* – люди, которые звонят, чтобы сообщить об экстренных ситуациях, и *машины скорой помощи*, работой которых управляет система.

Отметим, что *записи* являются важным выходом системы (в реальной жизни это отражает требование законодательства) и не менее важным средством для измерения «производительности».

Другие внешние потенциальные сущности системы, представленные на диаграмме, тоже обязательны для реальной системы, но для простоты мы исключим их из рассмотрения в данном примере.

Следующим шагом является определение внутренних функций системы.

Обычно вначале отображают функции для работы с каждой из внешних сущностей, получая, таким образом, минимальную функциональную декомпозицию системы. После этого отображают основные данные, которые должны транслироваться между этими функциями верхнего уровня (см. рис. 3.5).

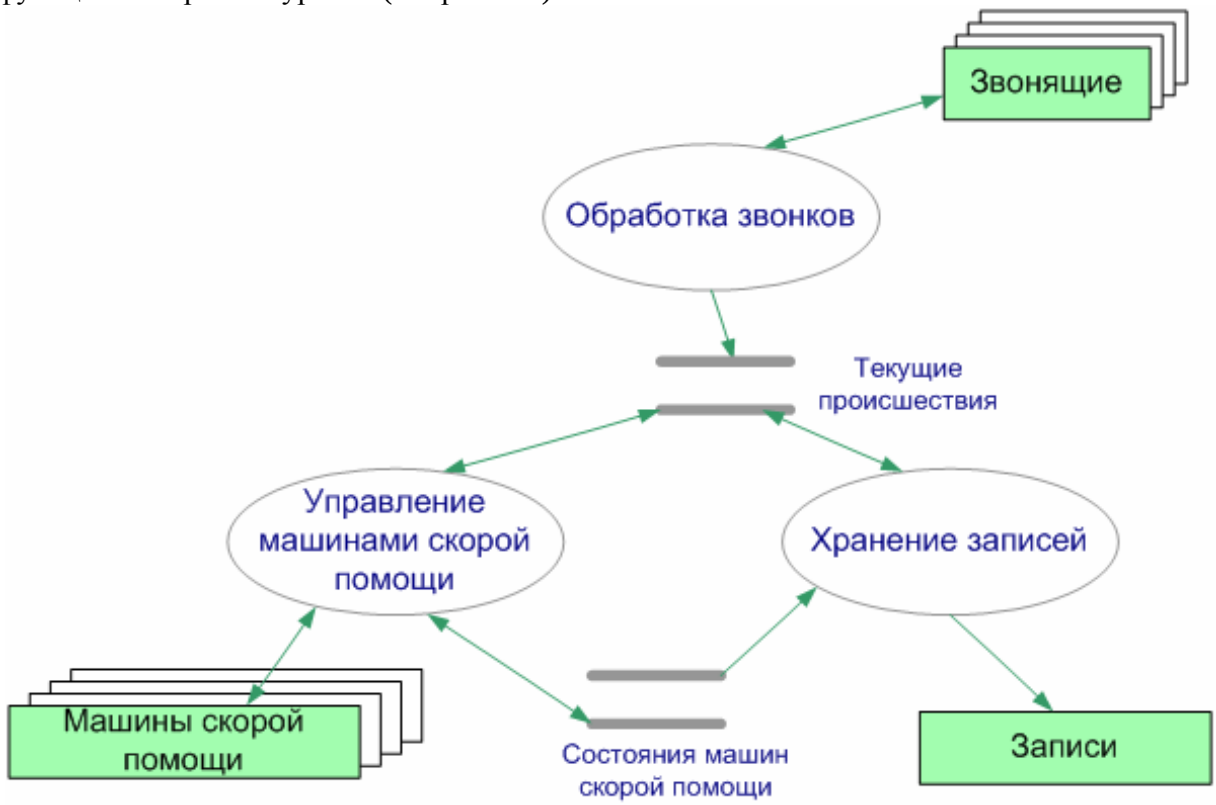


Рис. 3.5 Модель системы управления и контроля скорой помощи.

На следующем этапе функции верхнего уровня разбиваются на более мелкие составляющие функции, как это показано на рис. 3.6.

Следует заметить, что функциональная иерархия системы, отображаемая набором DFD диаграмм, может служить базой для получения и последующей структуризации системных требований. Так на рис. 3.7, как отмечено выше, представлена функциональная структура системы управления и контроля скорой помощи, полученная, в свою очередь, из диаграммы на рис. 3.6. А на рис. 3.7 показаны примеры требований полученных уже из этой структуры.

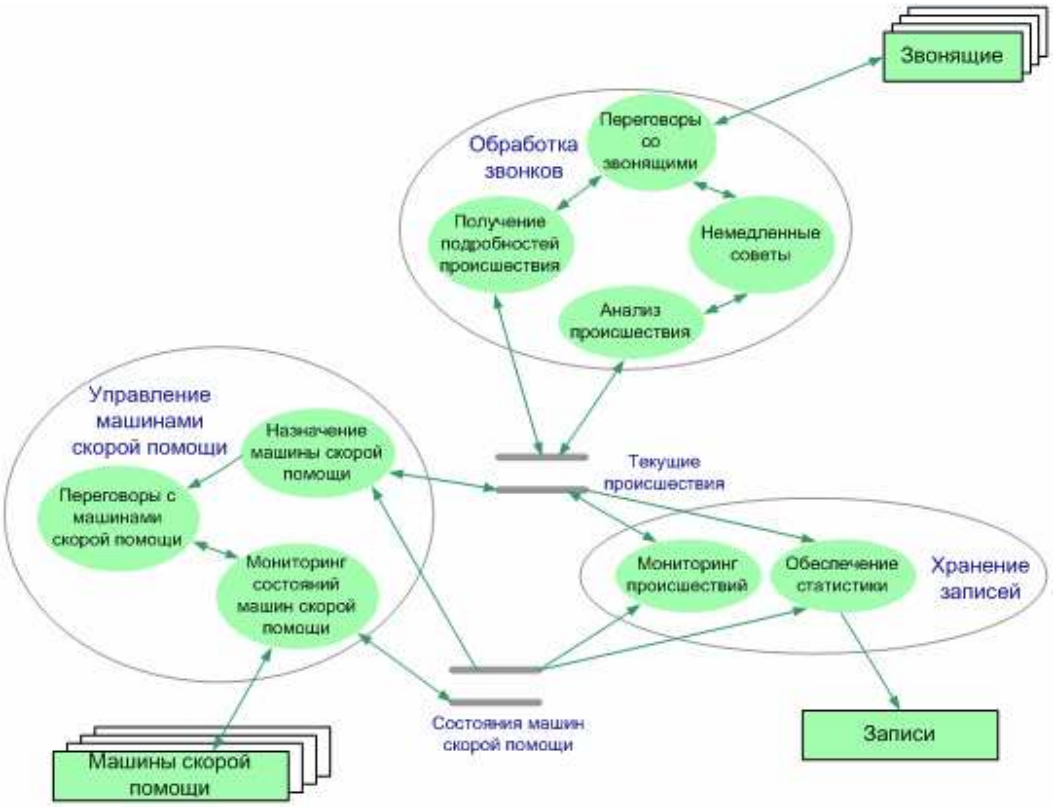


Рис. 3.6 Детализированная модель системы управления и контроля скорой помощи.

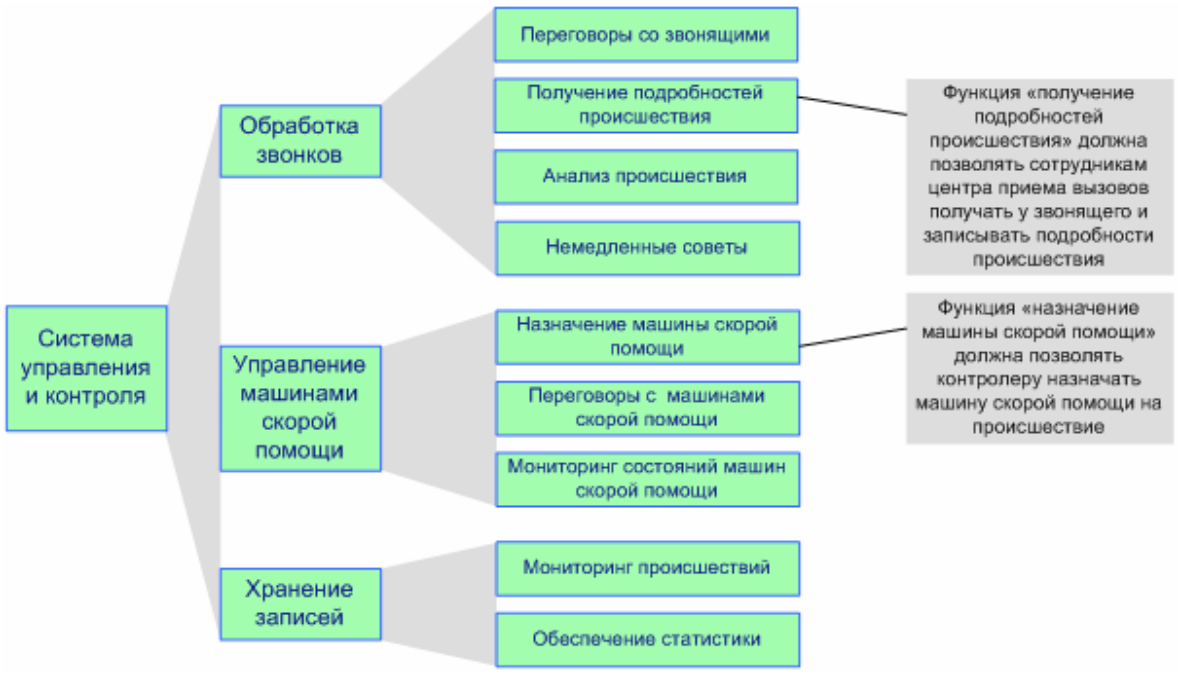


Рис. 3.7 Функциональная структура системы управления и контроля скорой помощи.

Представленная иерархия и интерфейсы могут хорошо описывать модель компонентов модели, но дают плохое представление о происходящих системных транзакциях (transactions), т.е. о тех действиях внутри системы, которые заключены между точкой входа и точкой выхода (см. рис. 3.8).

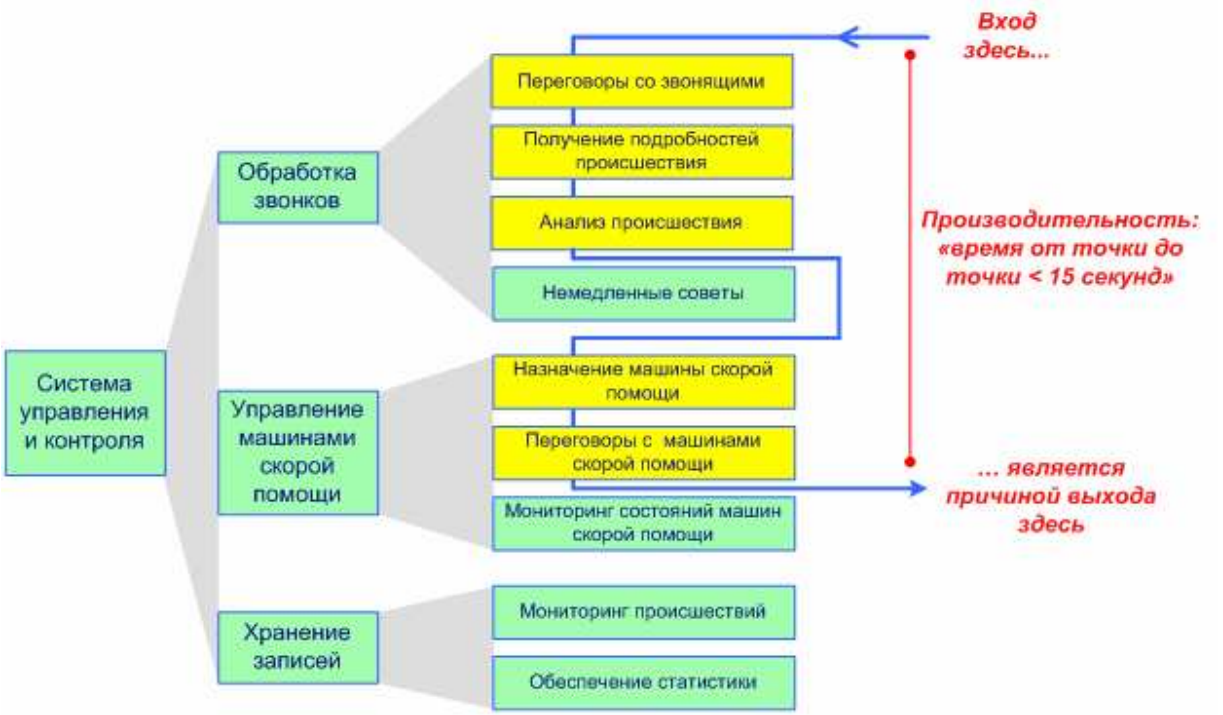


Рис. 3.8 Системные операции.

Отсюда возникает необходимость рассматривать системные транзакции с разных точек зрения – пути, который они проходят внутри системы; времени, которое требуется для их выполнения; ресурсов, которые они задействуют. Альтернативой динамическим моделям, показывающим пользовательские требования в действии и иллюстрирующим основные операции системы, является отображение системных транзакций на диаграммах потоков данных, как это сделано на рис. 3.9, где путь системной транзакции показан жирными стрелками.

Диаграммы потоков данных хорошо подходят для представления структур, но они недостаточно «аккуратны». Они гораздо менее точны, чем текстовое представление разрабатываемой системы. На диаграмме потоков данных линии связи могут быть неоднозначны, любое слово диаграммы может обобщать что угодно и иметь массу значений. Кроме этого, на диаграммах потоков данных нельзя корректно отображать ограничения.

Диаграммы потоков данных достаточно хорошо отображают функции и интерфейсы. Они также могут использоваться для определения системных транзакций типа «начало-конец» (end-to-end transactions), хотя и отражают их только косвенно. В идеале, конечно, было бы неплохо иметь возможность так просматривать диаграммы и «разворачивать» элементы структуры прямо на диаграмме, чтобы видеть контекст каждой диаграммы, каждого уровня декомпозиции. Кстати, некоторые CASE средства позволяют это делать.

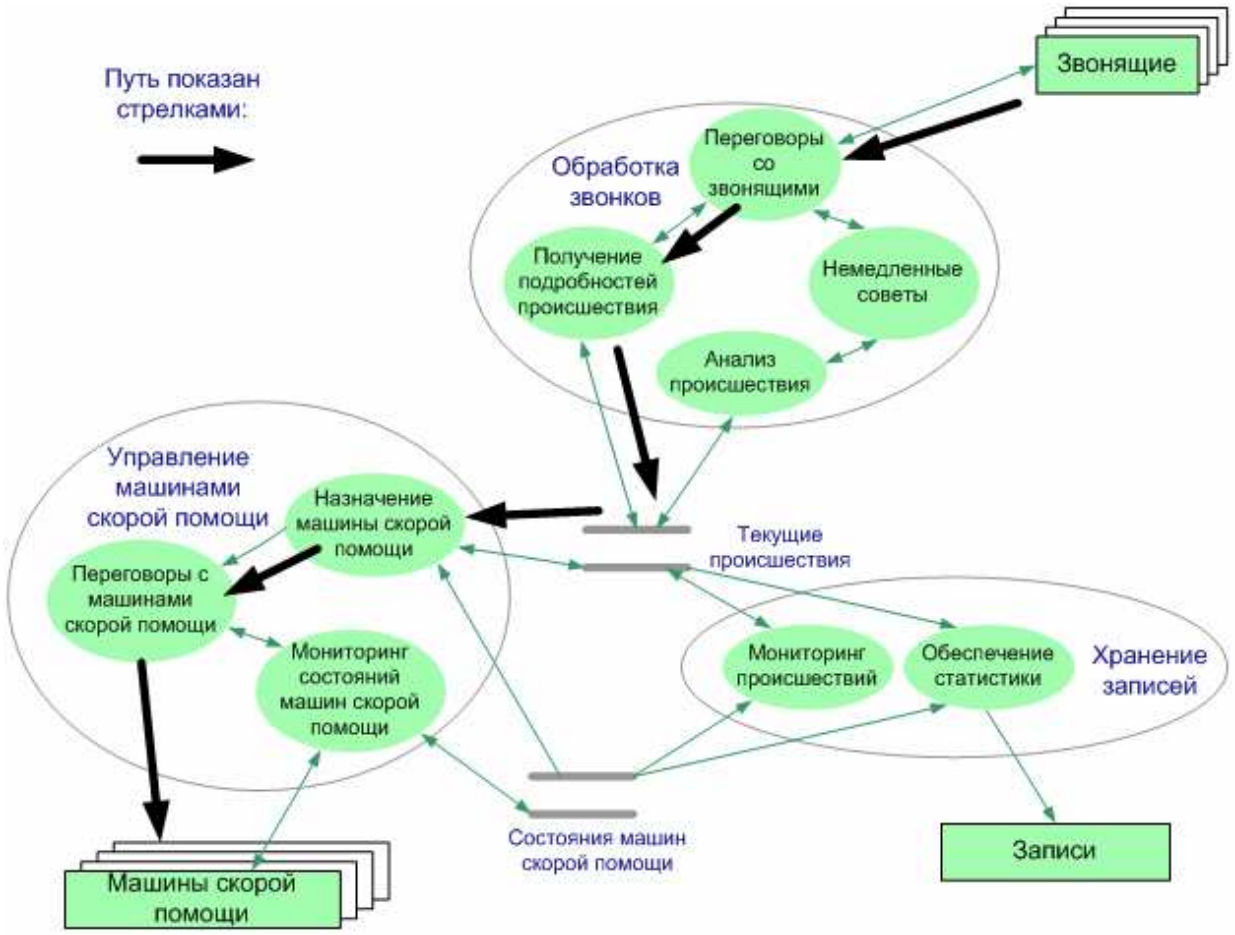


Рис. 3.9 Системные операции для системы управления и контроля скорой помощи.

Следует заметить, что рис. 3.6, на самом деле, является не совсем корректным с точки зрения договоренностей, принятых для отображения диаграмм потоков данных, поскольку на одной диаграмме показана и декомпозиция всей системы на несколько процессов, и показаны внешние сущности, с которыми взаимодействует система. Если точно следовать правилам построения DFD диаграмм, то внешние сущности должны присутствовать только на контекстной диаграмме и, следовательно, не должны присутствовать на этом уровне декомпозиции. Однако если убрать внешние сущности, то диаграмма станет менее понятной и связи к внешним сущностям зависнут в воздухе.
Вот почему неукоснительному следованию концептуально правильным идеалам, авторы книги предпочитают прагматический подход к использованию DFD.

Таким образом, диаграммы потоков данных:

- отображают в целом потоки данных и функциональность системы;
- определяют функции, потоки и хранилища данных;
- определяют интерфейсы между функциями;
- обеспечивают базу для получения системных требований;
- имеют инструментальные средства для работы с ними (специальное ПО);

- широко используются в разработке программного обеспечения;
- применимы для проектирования системам вообще (не только для ПО).

3.2.2 Диаграммы «сущность-связь»

Часто бывают ситуации, когда важной необходимостью является моделирование информации, которая хранится в системе, например, планы полетов, информация о наземных объектах и ориентирах, записи базы данных. В этом случае диаграммы сущность-связь (Entity-relationship diagrams или ERDs) как раз и предоставляют средства для моделирования сущностей системы и связей, существующих между ними.
Диаграммы сущность-связь были впервые разработаны Ченом (Chen) в 1976 г и в настоящее время существует множество нотаций ERD.
В таких диаграммах часто используется принятая терминология.

Сущность это объект, который может быть четко классифицирован и определен, например, заказчик, поставщик, деталь или продукт.
Свойство (или *атрибут)* это информация, которая описывает сущность.
Связь характеризуется множественностью, которая описывает тип связи между сущностями (один к одному, один ко многим, многие ко многим).
Подтипом называется подмножество объектов сущности (например, тип X является подтипом Y, если любой член множества X принадлежит к Y).

Диаграммы сущность-связь формируют модель, которая лишь частично описывает систему, поскольку определяет только сущности внутри системы и связи между ними. Эта модель является независимой по отношению к протекающим в системе процессам, которые требуют создания или использования информации. Однако такая модель является идеальным средством для абстрактного моделирования на этапе разработки системных требований.
На рис. 3.10 показан пример диаграммы сущность-связь для той же системы управления и контроля скорой помощи.

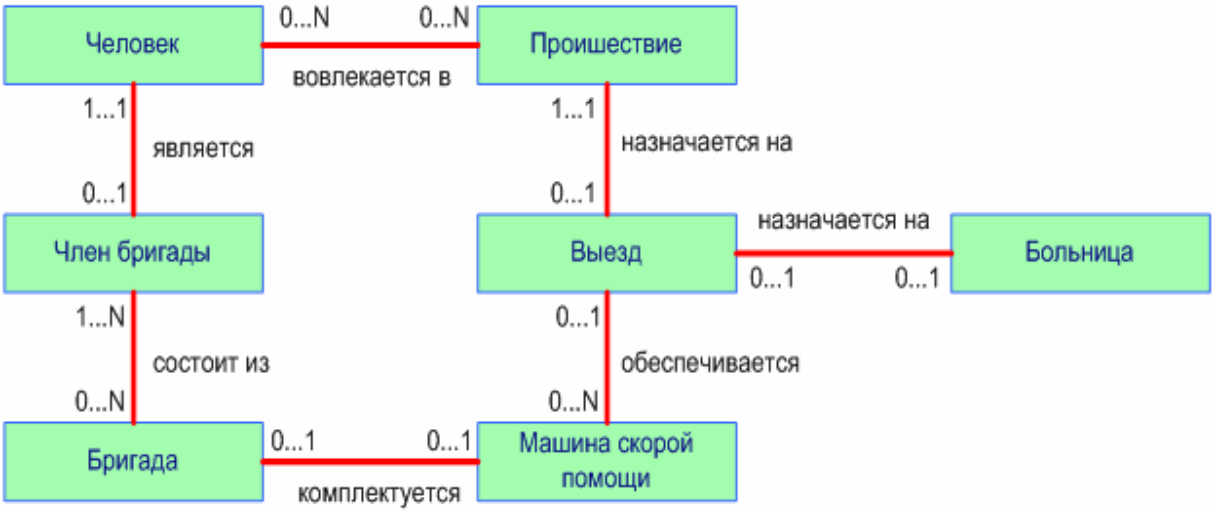


Рис. 3.10 Диаграмма сущность-связь для системы управления и контроля скорой помощи.

3.2.3 Диаграммы состояний

Для определения требований недостаточно иметь только функциональные модели и модели потоков данных. Необходимо также рассмотреть поведение системы в различных ситуациях, имея в виду, что при этом система может находиться в конечном числе возможных состояний.
Внешние события для системы могут служить триггерами, которые, срабатывая, переводят систему из одного состояния в другое. Для того чтобы рассмотреть систему в данном аспекте необходимо наглядно представить возможные состояния системы и ее реакцию на внешние события в разных состояниях. Обычно для этого используются диаграммы состояний (statecharts) Харела (Harel).

Диаграммы состояний обеспечивают описание поведения системы. В рамках одной диаграммы состояний может быть показана вся иерархия состояний системы и последовательность переходов системы из одного состояния в другое. Тем не менее, диаграммы состояний могут также использоваться и для описания систем, внутри которых возможно протекание параллельных процессов. На диаграмме состояний прямоугольником с закругленными углами обозначается состояние.

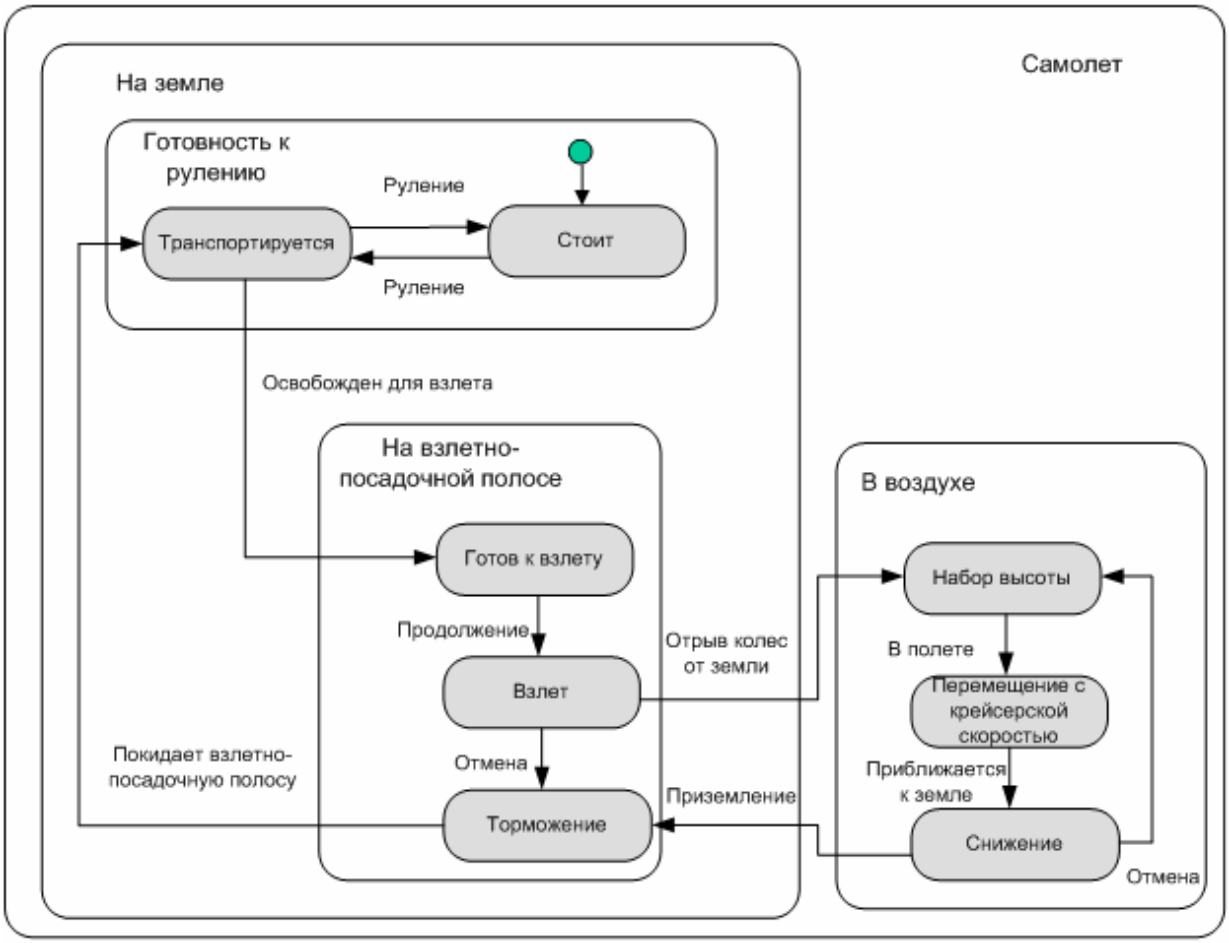


Рис. 3.11 Диаграмма состояний для полета самолета.

Иерархия состояний отображается путем включения одного состояния в другое и направленными стрелками, причем подписи к стрелкам являются описаниями событий, символизирующими переход между состояниями.

Поскольку диаграммы состояний описывают состояния системы, события и переходы между состояниями системы, то это делает их пригодными для моделирования систем.

На рис. 3.11 изображена диаграмма состояний, соответствующая полету самолета.
Два состояния верхнего уровня это – «на земле» и «в воздухе», между которыми определены переходы. Внутри состояния «в воздухе» существует три других независимых состояния, а внутри состояния «на земле» существуют состояния «готовность к рулению» и «на взлетно-посадочной полосе». При этом внутри состояний «готовность к рулению» и «на взлетно-посадочной полосе» также имеются собственные вложенные состояния.

Система переходит в состояние «в воздухе» когда колеса самолета отрываются от земли, а когда колеса касаются земли, система возвращается в состояние «на земле». Далее эти состояния детализируются в иерархическом порядке.

Диаграммы состояний имеют еще одну важную составляющую нотации – историю (history).
Здесь имеется в виду следующее.
Если имеется некое состояние, которое на диаграмме помечается специальным значком (H), то при возвращении системы в это состояние все вложенные состояния внутри этого H-состояния также считаются перешедшими в это первичное состояние.

3.2.4 Объектно-ориентированные подходы

Объектно-ориентированные подходы моделирования существенно отличаются от структурных подходов. Объекты представляют собой устойчивые и - будем надеяться на это - повторно используемые компоненты.

Объектно-ориентированные подходы направлены на максимизацию повторного использования инженерами объектов при разработке системных требований и спецификаций системы.

Таким образом целью объектно-ориентированного подхода являются:

- инкапсуляция (encapsulate), т.е. заключение внутрь *объектов* их поведения (состояния и событий), информации (данных) и операций;
- создание *устойчивых объектов*, которые могут быть использованы как для разработки требований, так и для разработки спецификаций системы;
- добавление информации путем большей детализации уже существующих *объектов*;
- создание новых объектов путем конкретизации существующих объектов, а не создание абсолютно новых.

Объектно-ориентированные подходы описывают поведение объектов и их взаимодействие между собой. Хотя порой при этом выполняется моделирование структуры объектов, но, на самом деле, это не является необходимым или даже желательным. Задачей аналитика является нахождение объектов, которые существуют наибольшее время, и моделирование поведения системы вокруг этих объектов. Такой поход позволяет получить ясное представление о поведении системы. Другая задача состоит в том, чтобы повторно

использовать существующие системные элементы, таким образом достигается значительное совершенствование последних.

Некоторые методологи настаивают на том, что спецификации системы (и даже ее реализация) должны являться следствием развития модели, полученной на этапе анализа. Реализовать такой подход достаточно непростая задача. Однако последовательное продвижение от анализа через спецификации к реализации системы с использованием объектно-ориентированных подходов гораздо проще и яснее, нежели чем выполнение того же самого с использованием других подходов. Следует заметить, что при использовании объектно-ориентированных подходов гораздо больше элементов, возникающих в процессе анализа, переходят непосредственно в стадию реализации, чем при применении структурных подходов анализа и проектирования. Это существенным образом помогает улучшить контролируемость связей внутри проектной документации и повысить удобство ее сопровождения.

Диаграммы классов (class diagrams)

Диаграмма классов является одной из основных нотаций объектно-ориентированного анализа и проектирования.

Впервые объектно-ориентированное направление в разработке и дизайне появилось благодаря возможностям компьютерной имитации. Главный принцип компьютерной имитации состоит в том, что программа должна моделировать реальный мир. Самый естественный путь к этому заключается в том, чтобы компьютерная программа оперировала *объектами*, которые являются отражением сущностей реального мира и которые моделируют их действия и отражают информацию (свойства) которой они обладают.

Например, в банковской системе вместо того, чтобы иметь файл с информацией о счете и отдельную бухгалтерскую программу, можно создать объект *счет*, который будет содержать информацию о *балансе* и *лимите на превышение кредита*, а также будет иметь связи с другими объектами, например, с объектом *владелец счета*. Эти объекты также будут иметь *операции* (или *методы*), которые могут выполняться со счетом, например, *проверить баланс*, *пополнить счет*, *снять со счета*.

На рис. 3.12 представлен пример диаграмм классов (или объектов).

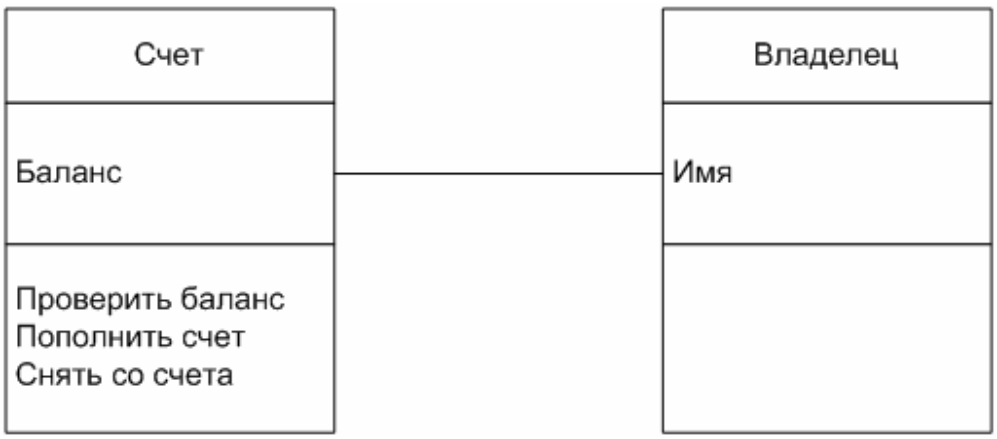


Рис. 3.12 Диаграмма классов.

Основной причиной появления этого подхода было желание сделать разработку программного обеспечения более близкой к моделированию, а, следовательно, и более естественной. К сожалению, на реализацию любой хорошей идеи практика оказывает свое влияние, поэтому лишь совсем небольшое количество объектно-ориентированных программных систем могут быть признанными действительно полным отражением реального мира. Тем не менее, это никак не умаляет достоинств самого метода.
На диаграмме классов изображается информация о классах объектов и связях между ними. Во многом диаграмма классов похожа на диаграмму сущность-связь. Так же как на диаграмме сущность-связь, диаграмма классов показывает, как объекты определенных классов связаны с другими объектами этого же класса, или других классов.

Основные добавленные элементы информации это:

- операции (методы);
- концепция обобщения (generalization);
- атрибуты внутри объекта.

Прецеденты (Use cases)

Прецеденты описывают взаимодействие, которое может иметь место между системой и ее пользователями (актерами; в иностранной литературе - actors).
Прецеденты изображаются овалами, так же как и процессы на диаграммах потоков данных (DFD). На диаграммах прецедентов изображаются прецеденты, актеры и связи между ними. При этом каждый прецедент определяет функциональное требование к системе.

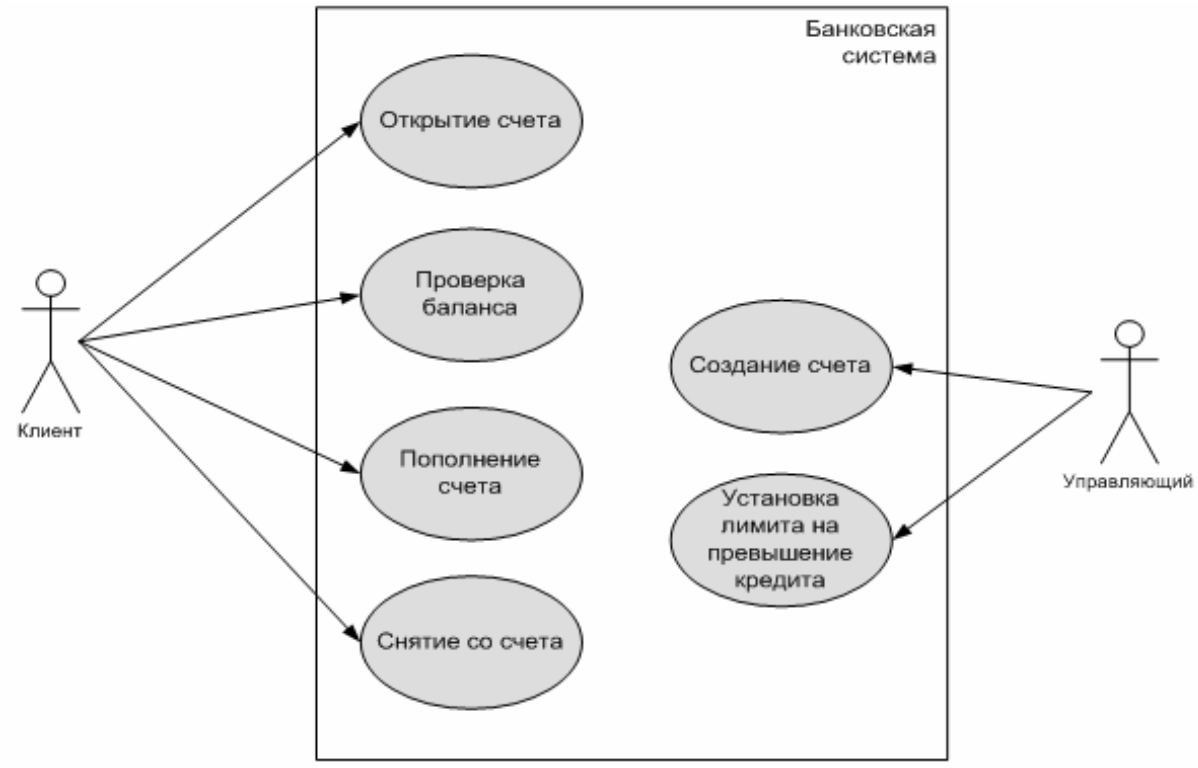


Рис. 3.13 Диаграмма прецедентов для банковской системы.

Не смотря на то, что на диаграммах актер изображается схематичной фигуркой человечка, это не обязательно должен быть человек. На самом деле здесь более важно то, что актеры символизируют *роли*. При этом каждый актер должен быть связан как минимум с одним прецедентом.

Рамки системы обозначаются на диаграммах в виде прямоугольника, в углу которого приводится название системы. Обычно для каждой диаграммы прецедентов разрабатывается и текстовое описание. Это является существенным и полезным аспектом.

На рис. 3.13 изображена диаграмма прецедентов для банковской системы.

3.3 Методы

Метод - это нечто носящее более рекомендующий характер, нежели просто моделирование. Метод предписывает, что делать и в каком порядке делать.

Методы используют различные представления - от текстового описания и диаграмм, до строгой математики. Метод предписывает, когда и где использовать эти представления.

Те методы, которые в основном ориентированы на использование диаграмм, обычно относятся к «структурным методам». Методы, которые используют объектные представления, обычно относятся к «объектно-ориентированным методам». Методы же, которые в основном используют математику, относятся к «формальным методам».

Основной целью представлений, используемых тем или иным методом, является сбор информации. Синтаксические правила разработки диаграмм определяют набор основных концепций, помогающих собирать информацию на диаграмме.

Как мы уже отмечали в предыдущих разделах данной главы, существует достаточно большое количество (графических) представлений, используемых для системного моделирования.

Основными *методами*, которые можно отметить, являются (по именам создателей) – ДеМарко (DeMarco, 1978), Иордон (Yourdon, 1990), Румбах (Rumbaugh, 1991), Шлер и Меллор (Shlaer, Mellor, 1998). Остальные методы, которые здесь также можно было бы перечислить, являются модификацией названных, в которых лишь изменен немного порядок действий и добавлены некоторые незначительные улучшения.

Стоит отметить, что сходств между этими методами гораздо больше, чем различий.

3.3.1 Методы перспектив

Любой подход, основанный на использовании перспектив (различных точек зрения), базируется на том, что требования к системе не должны рассматриваться только с одной стороны (точки зрения). Любой такой подход должен придерживаться того правила, что сбор и организация требований должны основываться именно на различных точках зрения (перспективах). По существу предлагается два основных типа перспектив:

- точки зрения заинтересованных сторон (stakeholders);
- точки зрения экспертов, обладающих знаниями в предметной области и организации.

Роль и влияние точки зрения заинтересованных сторон для проекта вполне понятна и не вызывает вопросов у разработчиков требований. При этом точки зрения экспертов,

имеющих знания в предметной области, могут отражать различные аспекты безопасности, маркетинга, систем хранения и обработки данных, государственного регулирования, стандартов и т.д. Другими словами, люди, обладающие знаниями в этих областях, могут быть не связаны ни с одной из заинтересованных сторон, однако и эту информацию, исходящую из большого количества различных источников, необходимо обязательно учитывать.

В последующих разделах рассматриваются три различных метода, основанных на использования перспектив.

Контролируемое формулирование требований (Controlled Requirements Expression или *CORE)*

Метод контролируемого формулирования требований был изначально разработан в процессе работы по анализу требований для министерства обороны Великобритании.

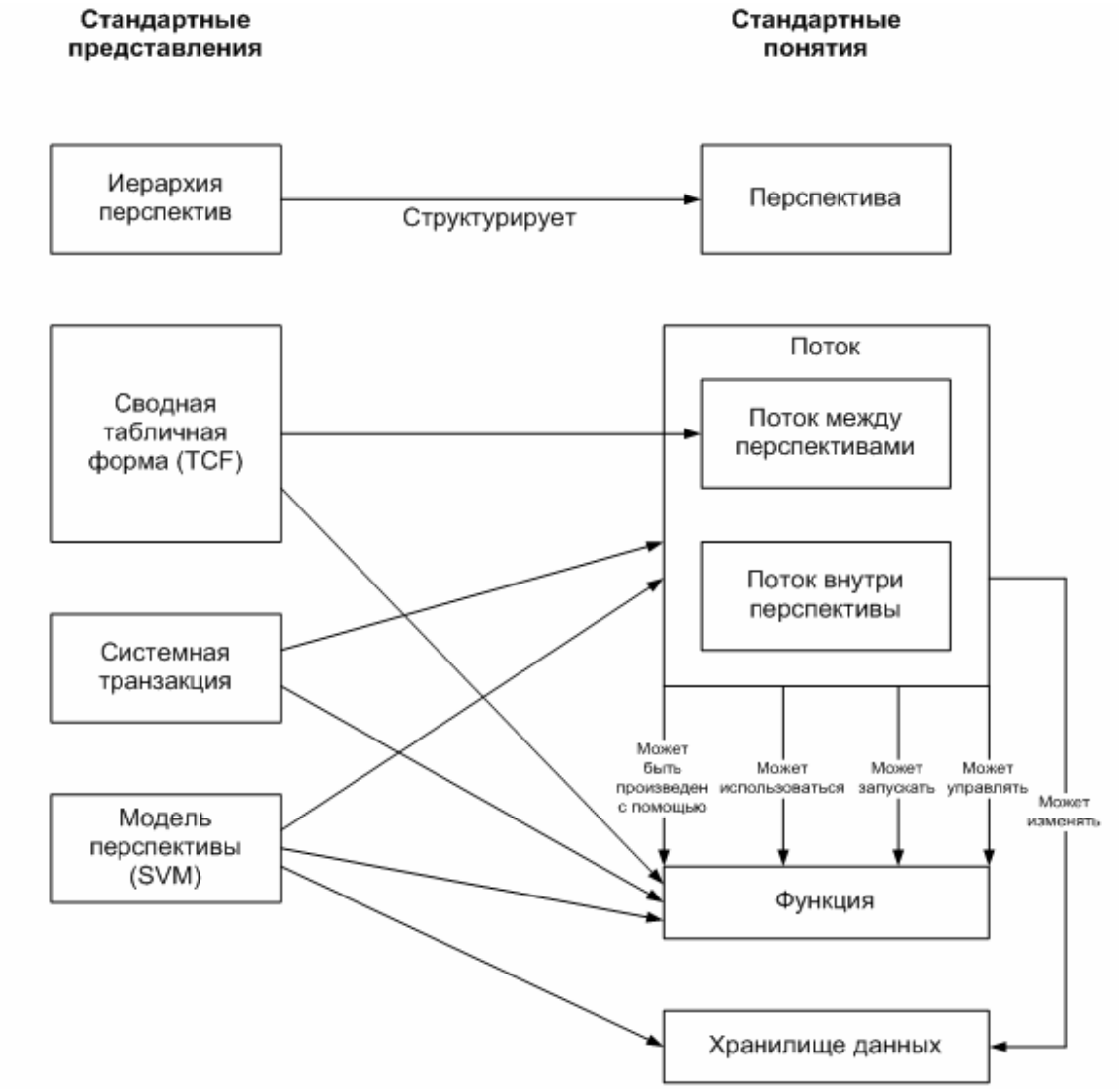


Рис. 3.14 Представления и понятия метода контролируемого формулирования требований.

Дело в том, что применяемые в то время методы зачастую с самого начала, т.е. до оценки возможных потенциальных решений, сразу предполагали формулирование контекста решения проблемы, а никак не попытку выяснения сути самой проблемы. Данный метод был разработан специально, чтобы исправить эту ситуацию.
На рис. 3.14 изображены основные понятия и представления, использующиеся в методе.

Основным понятием метода является перспектива (точка зрения) и связанное с ней представление, известное как иерархия перспектив. В качестве перспективы может быть выбрана точка зрения на разрабатываемую систему отдельного человека, роли или организации [Такой подход был использован Дарке и Шанксом (Darke, Shanks, 1997) как основа для анализа точек зрения пользователей]. На уровне системных требований перспективы могут также отражать предназначение самой системы, а также ее частей или подсистем, входящих в окружение разрабатываемой системы и оказывающих влияние на ее функционал. Перспективы образуют иерархию, которая используется для определения границ системы и помогает процессу анализа.

В качестве иллюстрации вышесказанного, на рис. 3.15 представлен предварительный перечень возможных перспектив для системы контроля за торможением самолета (Aircraft brake and control system – ABCS), полученный в результате мозгового штурма.

Рис. 3.15 Предварительный перечень перспектив для системы контроля торможением самолета.

Имея предварительный перечень перспектив, его упорядочили в виде иерархической структуры, сгруппировав связанных между собой кандидатов. Путем последовательных итераций вокруг связанных групп прорисовывались границы. И это повторялось до тех пор, пока все кандидаты не были включены в иерархическую структуру.

На рис. 3.16 представлена часть иерархии перспектив для системы контроля за торможением самолета.

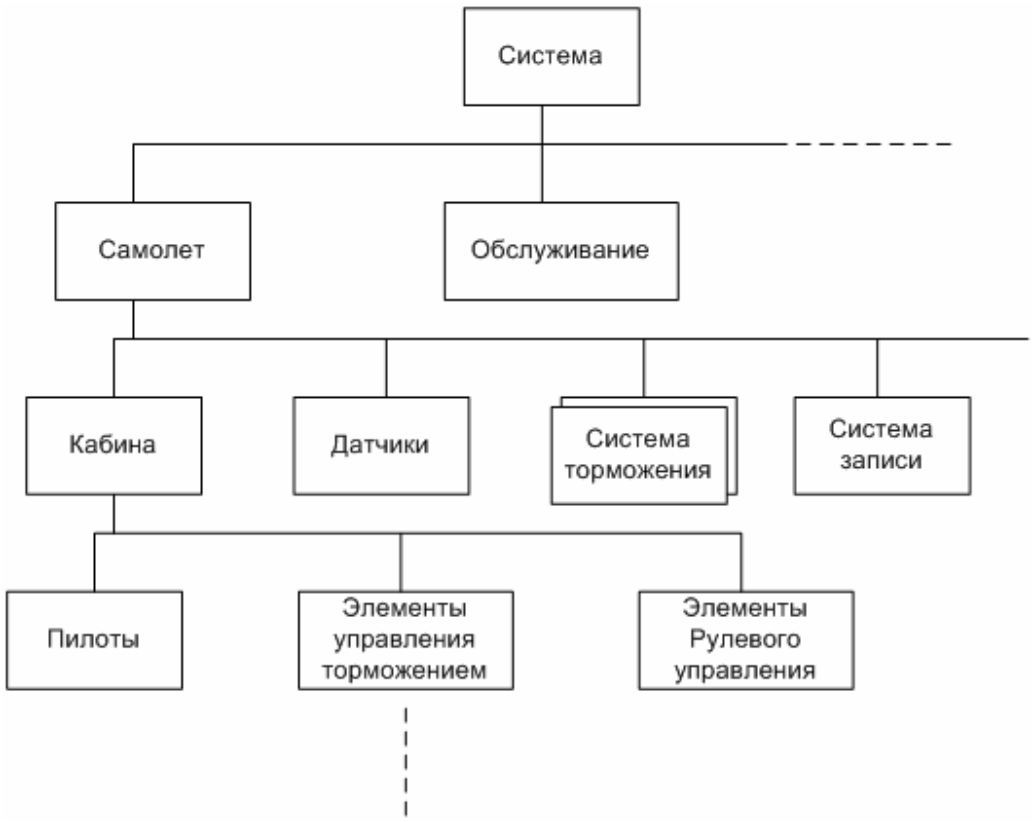


Рис. 3.16 Пример иерархии.

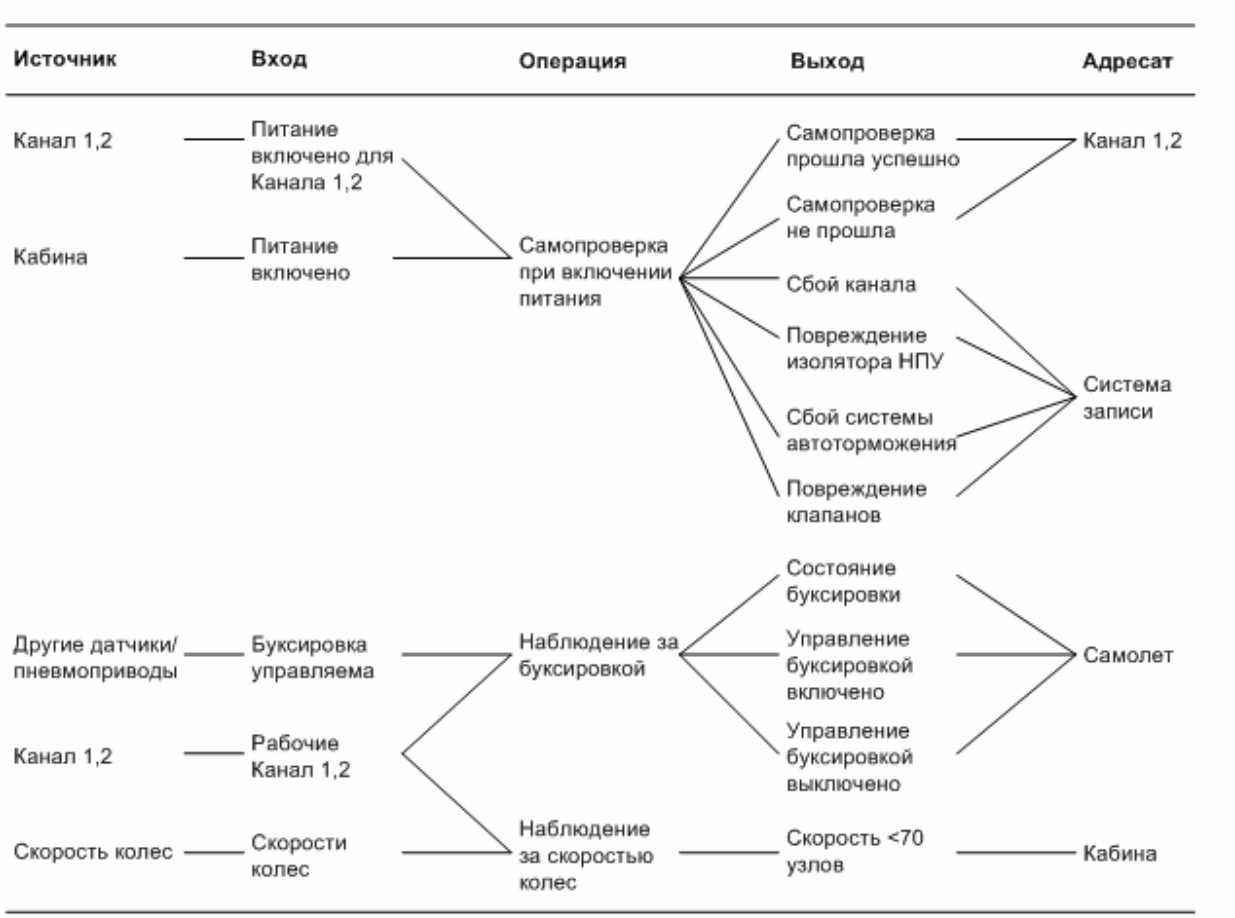
Метод контролируемого формулирования требований предусматривает, что любая операция, выполняемая со стороны перспективы, должна быть четко определена. Каждая операция может использовать или производить информацию или другие элементы, характерные для рассматриваемой системы (например, товары). Вся информация, полученная в результате такого анализа, заносится в сводную табличную форму (TCF = Tabular Collection Form), возможный образец которой показан в таблице 3.1.

Таблица 3.1 Сводная табличная форма

Источник	Вход	Операция	Выход	Адресат
Перспектива, порождающая вход	Название входного элемента	Операция, выполняемая над одним или несколькими входным элементами для получения требуемого выхода	Название(-я) выходного(-ых) элемента(-ов), производимого(-ых) операцией	Перспектива, которой посылаются выходные элементы

В качестве иллюстрирующего примера в таблице 3.2 представлена часть сводной табличной формы для системы управления за торможением самолета.

Таблица 3.2 Пример сводной табличной формы (TCF)



Линии, связывающие между собой соседние столбцы, обозначают существование потоков.

Затем каждая перспектива анализируется таким образом, что сводная табличная форма (TCF) на каждом уровне иерархии перспектив выглядит и проверяется как группа для того, чтобы быть уверенным, что а) входы, которые каждая перспектива ожидает получить, выходят из источника перспективы и что б) выходы, которые генерируются каждой транзакцией, ожидаются (к получению) перспективой\-ами, которые предназначены для их получения.

Дальнейший анализ заключается в поочередной разработке более подробных моделей потоков данных для каждой перспективы. При этом отправной точкой для создания моделей перспектив служит информация, занесенная в сводную табличную форму (SVMs = Single Viewpoint Models). Модели перспектив описывают дополнительно потоки, находящиеся «внутри» перспективы, а также хранилища данных. Модель перспективы также описывает то, как операции перспективы контролируются и управляются в зависимости от потоков в других операциях (и реагируют на события в них).

Таким образом, анализ осуществляется сверху вниз через все уровни иерархии перспектив. Но, двигаясь сверху вниз, зачастую бывает трудно определить, когда нужно остановиться и куда в результате может привести анализ. Поэтому подход заключается в

том, чтобы сначала определить все перспективы, а затем использовать их структуру для контроля и последующего анализа. Как раз это-то и помогает решить основную проблему, связанную с анализом, основанным на рассмотрении потоков данных. Именно поэтому метод называется «Контролируемое формулирование требований».

Другое основное понятие метода CORE это *системная транзакция*.

Системная транзакция это путь через систему от одного или нескольких входов, потоков данных или событий до одного или нескольких определенных выходных потоков или событий. Системные транзакции описывают то, как система должна функционировать. Они обеспечивают совершенно другой взгляд на систему, нежели чем иерархия перспектив. Системные транзакции обеспечивают смысловое наполнение для обсуждения нефункциональных требований.

Техника структурного анализа и проектирования (Structured Analysis and Design Technique или SADT)

Метод техники структурного анализа и проектирования (Structured Analysis and Design Technique или SADT) берет начало из работ Росса (Ross), посвященных структурному анализу, которые он проводил в 70-х годах.

Этот метод основан на применении диаграмм.

Основой метода является использование иерархической декомпозиции; когда решение проблемы находится путем последовательного проектирования модулей и последующего уточнения их архитектуры.

Основной элемент графической нотации метода является прямоугольник, который обозначает действие (на диаграммах действий), или данные (на диаграммах данных).

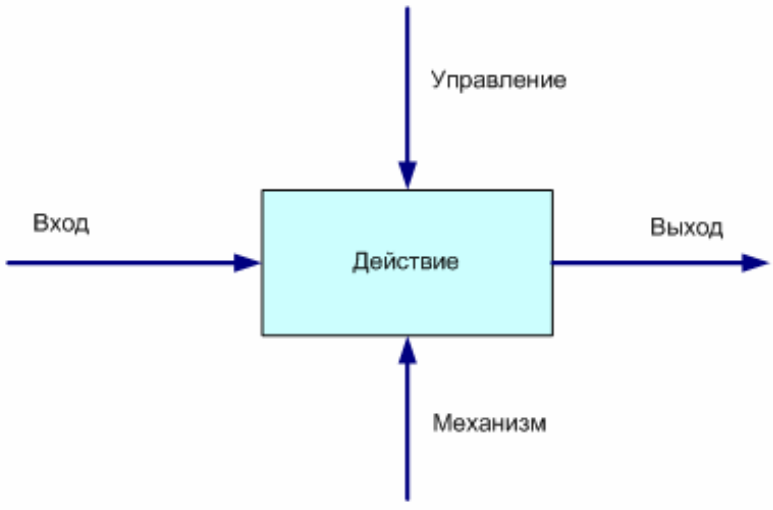


Рис. 3.17 Основные элементы техники структурного анализа и проектирования.

Прямоугольники соединяются стрелками, которые обозначают или данные, необходимые действию либо производимые им (в случае если прямоугольником обозначено действие на диаграмме действий), или же стрелки обозначают процессы, использующие или

производящие данные (естественно, что в этом случае на диаграммах данных прямоугольником обозначаются данные).

На рис. 3.17 показаны четыре основных типа стрелок. Тип стрелки зависит от того, с какой стороны она соединяется с прямоугольником (подходит к нему):

- *Входящие* стрелки это те, которые соединяются с прямоугольником с левой стороны, и представляют собой данные, которые получает прямоугольник, обозначающий действие.
- *Исходящие* стрелки это те, которые соединяются с прямоугольником с правой стороны, и представляют собой данные, которые производит действие, обозначенное этим прямоугольником. Другими словами входящие данные преобразуются действием в производные данные.
- *Управляющие* стрелки соединяются с прямоугольником сверху и определяют способ преобразования входящей информации.
- Стрелки *механизма* подходят к прямоугольнику снизу и, по сути, обозначают способ, с помощью которого действие может использовать внешние механизмы для преобразования входящих данных, например, определенный алгоритм или ресурсы.

Любая диаграмма в этом методе состоит из ряда прямоугольников, связанных стрелками. Таким образом, любая проблема детализируется и уточняется путем последовательной декомпозиции каждого действия и создания иерархических диаграмм, как это показано на рис. 3.18.

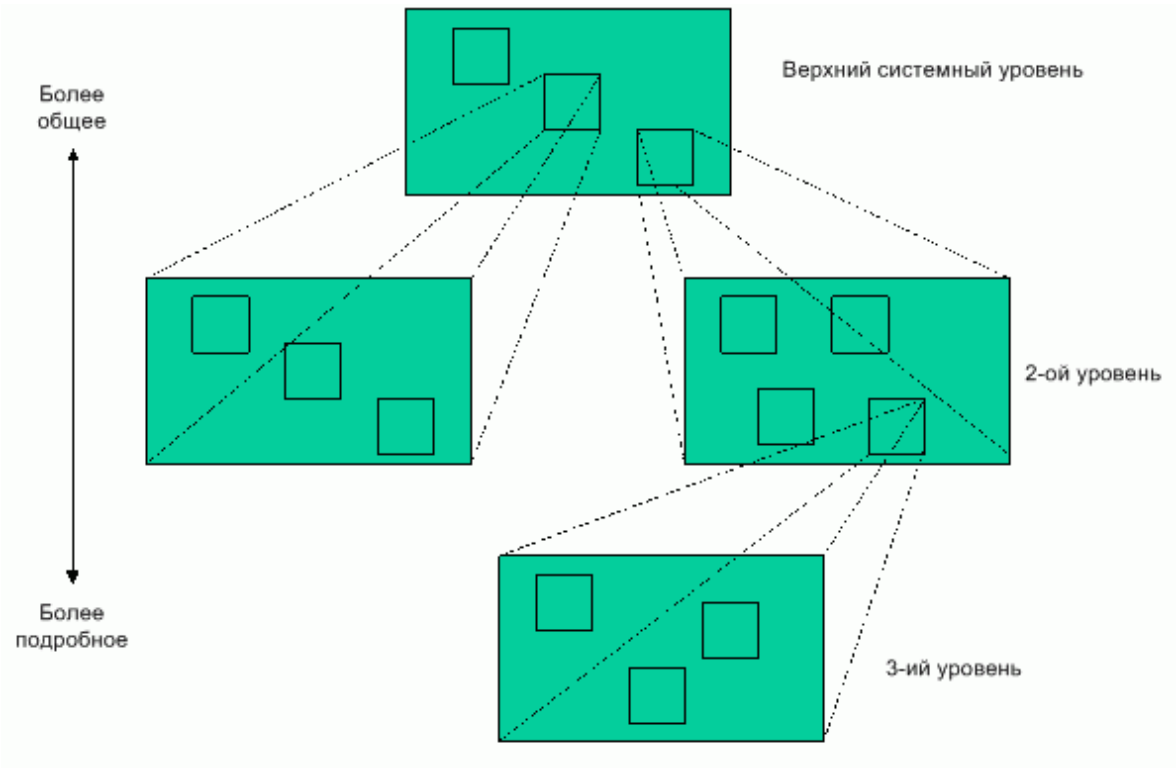


Рис. 3.18 Проведение декомпозиции с использованием техники структурного анализа и проектирования.

На рис. 3.19 представлен пример диаграммы действий для системы контроля торможением самолета. Декомпозиция продолжается до тех пор, пока не будет получено достаточно подробное описание для начала разработки архитектуры решения.

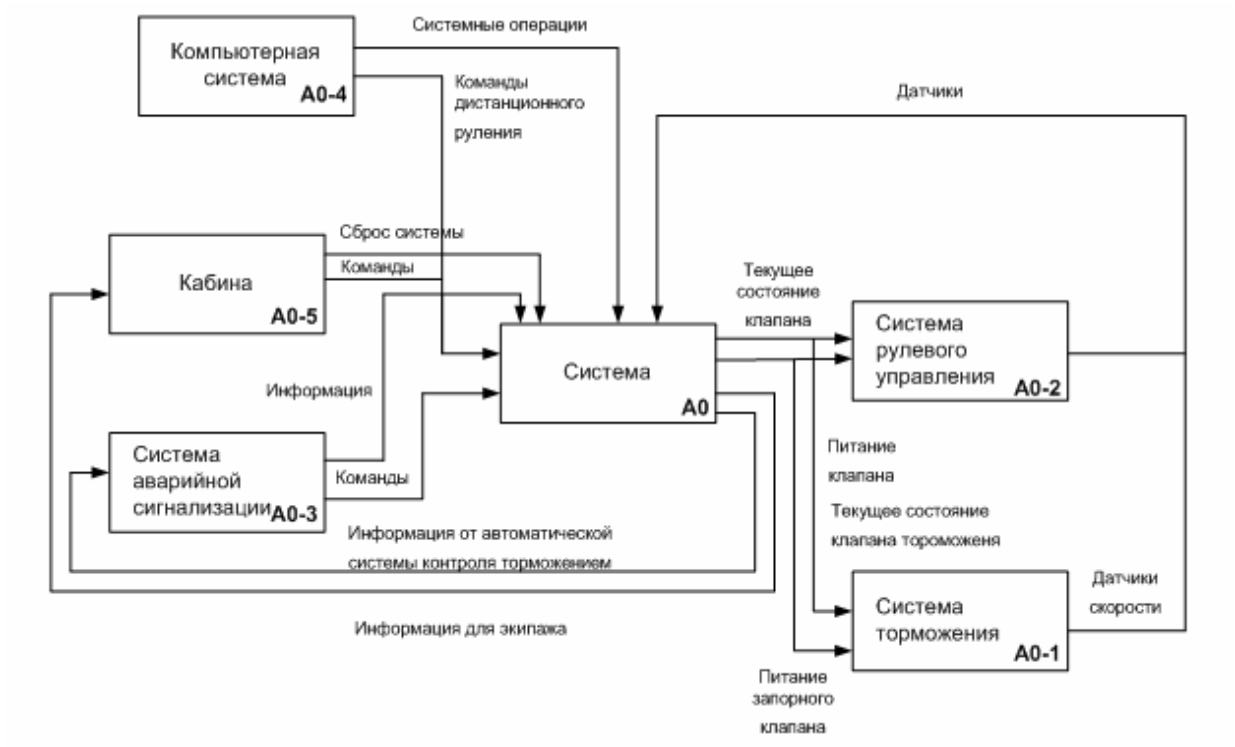


Рис. 3.19 Пример использования техники структурного анализа и проектирования (SADT).

Определение требований, основанное на перспективах (Viewpoint-oriented Requirement Definition или VORD)

Определение требований, основанное на перспективах [Котонья (Kotonya) и Саммервилль (Sommerville), 1996] это метод, базирующийся на парадигме рассмотрения различных перспектив (точек зрения).

Метод использует «сервис-ориентированную» модель, в которой перспектива может рассматриваться в качестве «клиента», если эту модель сравнивать с системой типа «клиент-сервер». Модель ориентирована на предоставление сервисов клиентам, точки зрения (перспективы) которых принимаются во внимание при разработке системы.

Концепция метода заключается в том, что «клиент» (= перспектива) получает от системы сервис, а в свою очередь, обеспечивает для системы поток управления.

Такой подход делает этот метод особенно полезным для проектирования интерактивных систем.

В методе VORD рассматривается два типа перспектив – **прямые** и **косвенные**:

- Перспективы, которые являются **прямыми,** получают сервисы от системы и посылают системе управляющие сигналы и данные.

- Перспективы (точки зрения), которые являются **косвенными**, не взаимодействуют с системой напрямую, но, тем не менее, «имеют свой интерес» как минимум в одном из сервисов, предоставляемых системой.

Косвенных перспектив может быть великое множество.
В качестве примера можно привести системных инженеров, которых интересуют аспекты, связанные с разработкой системы и ее дальнейшим обслуживанием, или некие внешние стороны, чьи точки зрения могут быть связаны с окружением системы и вопросами безопасности, связанными с ее эксплуатацией.

Метод предусматривает три основных неоднократно применяемых шага:

- определение перспектив и их структуры;
- документирование перспектив;
- разработка требований и их детализация на основе перспектив.

На рис. 3.20 показана графическая нотация, используемая этим методом.
Перспектива представлена с помощью прямоугольника, который содержит идентификатор, тип и название. Атрибуты (или свойства) перспективы также обозначаются прямоугольниками, каждый из которых содержит идентификатор атрибута и его название. Прямоугольники атрибутов присоединяются с левой стороны к линии, спускающейся вниз из прямоугольника перспективы.

Рис. 3.20 Графическая нотация для моделирования перспектив.

Использование метода предполагает последовательное рассмотрение и выделение (идентификацию) необходимых перспектив. Вначале предлагается рассматривать общий набор абстрактных перспектив (см. рис. 3.21), который можно считать стартовой точкой в процессе идентификации (по соглашению принятому в VORD, прямые перспективы обозначаются белыми прямоугольниками, косвенные – серыми). Затем этот набор сокращается удалением ненужных перспектив, которые не являются существенными с точки зрения рассматриваемой проблемы. А уж далее определяются все заинтересованные лица и перспективы, относящиеся к другим системам, взаимодействующим с проектируемой, а также все непосредственные пользователи системы. И, в конце концов,

для каждой из идентифицированных косвенных перспектив определяется, с чем они могут быть связаны (ассоциированы).

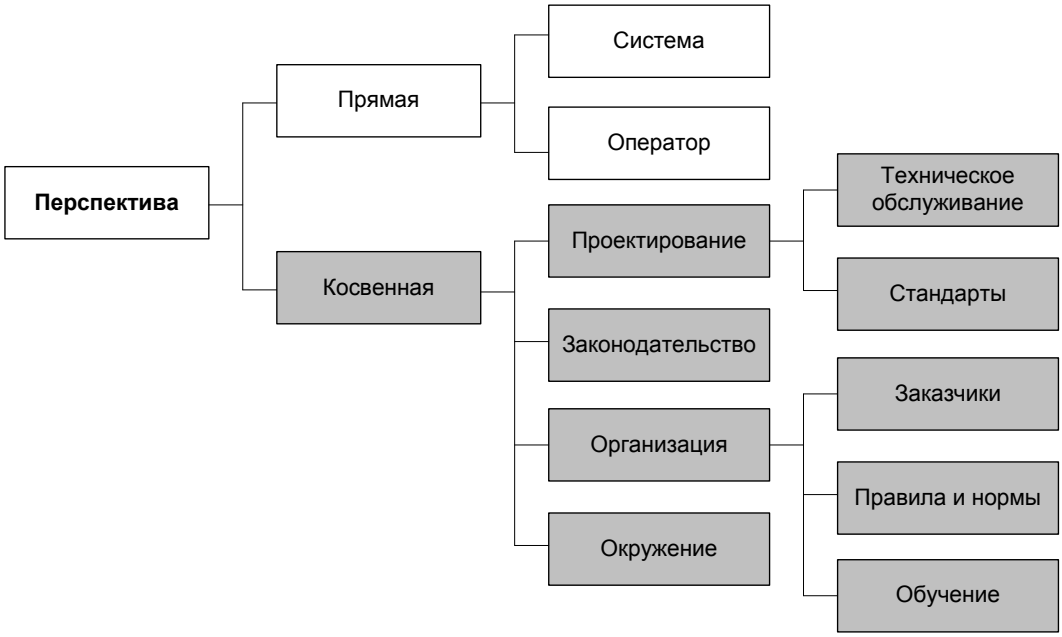


Рис. 3.21 Классы перспектив.

На рис. 3.22 представлены перспективы для системы автоматической оплаты парковки.

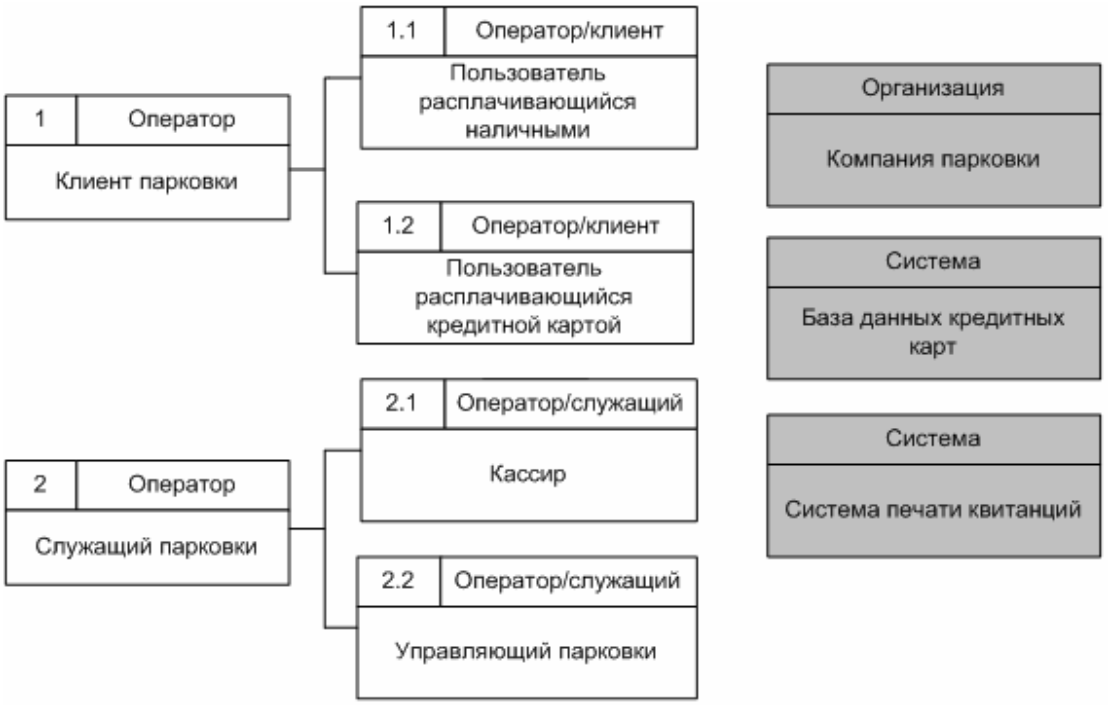


Рис. 3.22 Перспективы для системы автоматической оплаты парковки.

Как видно из рисунка, «Пользователь, расплачивающийся наличными», и «Пользователь, расплачивающийся кредитной картой», являются детализациями перспективы «Клиент парковки»; а «Кассир» и «Управляющий парковки» являются детализациями перспективы «Служащий парковки». Перспектива «Система печати квитанций» представляет собой базу данных организации, отвечающей за прием оплаты и представление квитанций. «База данных кредитных карт» является внешней перспективой и представляет систему, хранящую информацию о кредитных картах клиентов.

Следующим шагом метода VORD является документирование требований для каждой из перспектив. В качестве примера того, как это делается, в таблице 3.3 приведены начальные требования для перспективы «Клиент парковки».

Таблица 3.3 Требования для точки зрения клиента автомобильной парковки.

Требование перспективы				
Идентификатор	**Название**		**Описание**	**Тип**
1	Клиент	1.1	Обеспечить возможность для выписки квитанций на соответствующую оплату за время стоянки	sv
1.1	Пользователь, расплачивающийся наличными			
1.2	Пользователь, расплачивающийся кредитной картой	1.2.1	Обеспечить возможность оплаты действительной кредитной картой	sv
		1.2.2	Обеспечить печать квитанции для клиента	sv
		1.2.3	Функция печати квитанций должна быть доступна в 99% процентах обращений	nf
		1.2.4	Время отклика сервиса печати квитанций не должно превышать 30 секунд	nf

В таблице в графе тип требований аббревиатура sv обозначает сервис (service), а nf обозначает нефункциональное требование (non-functional requirement). Как упоминалось ранее, каждая из перспектив может иметь свойства, которые определяют характеристики перспективы с точки зрения проблемной области. Эти свойства являются важными, поскольку обеспечивают дополнительную информацию о среде, в которой должна функционировать разрабатываемая система (см. рис. 3.23).

Системное поведение моделируется при помощи сценариев событий, которые описывают взаимодействие системы с внешним окружением, обеспечивая тем самым способ описания комплексного взаимодействия между различными пользователями (точки зрения которых обозначены перспективами) и системой.

Заключительной стадией метода VORD является разработка документа требований в соответствии с промышленными стандартами на основе полученных результатов анализа.

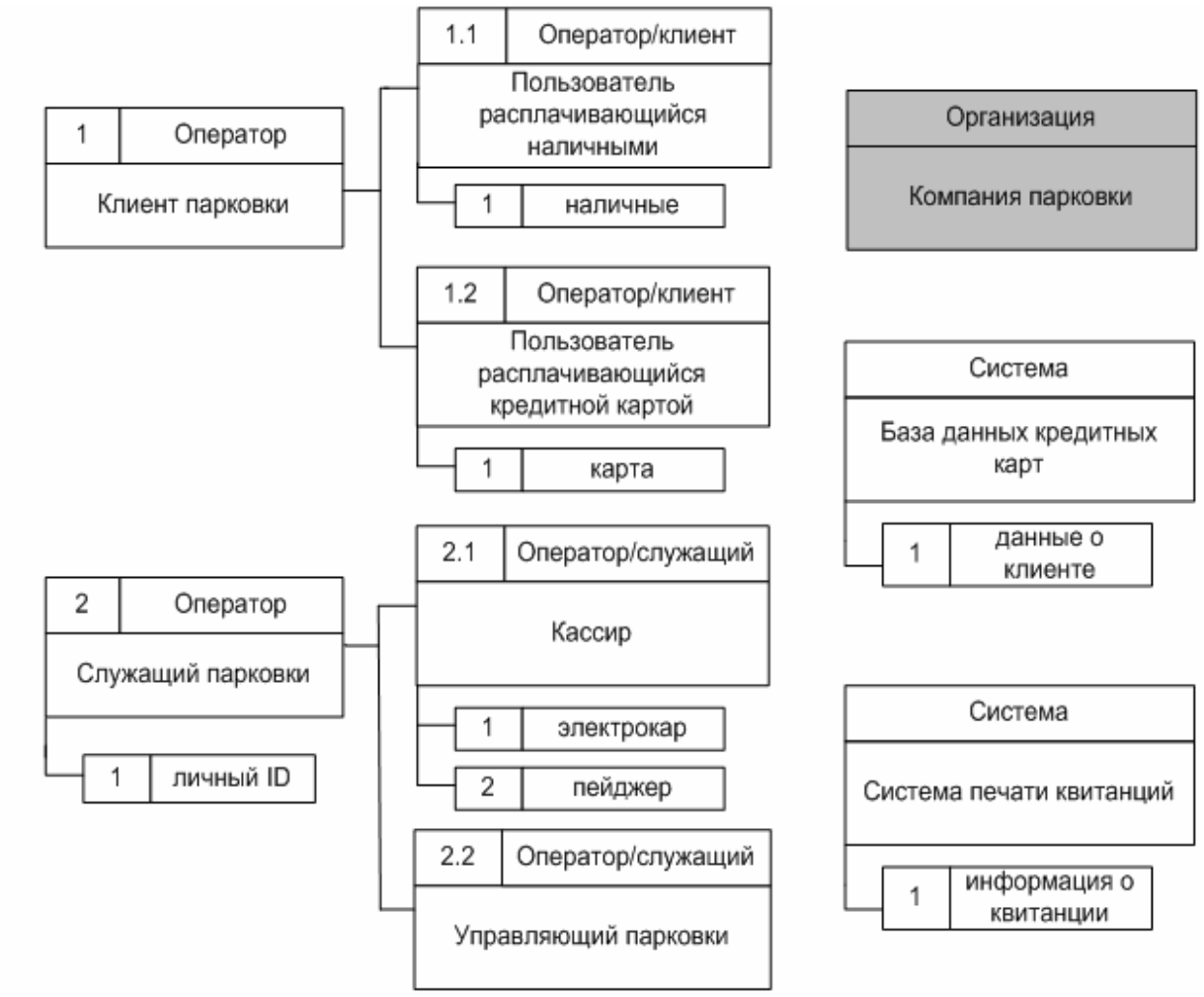


Рис. 3.23 Атрибуты перспектив.

3.3.2 Объектно-ориентированные методы

В конце 80-х и начале 90-х годов XX века возникли различные объектно-ориентированные (О-О) методы, предлагающие подходы объектно-ориентированного анализа и разработки. Ранние О-О методы возникли в тех компаниях, для которых было очень важно обеспечить небольшое время выхода продукта на рынок и высокую устойчивость продукта к изменениям. В числе таких компаний были телекоммуникационные и финансовые, а позднее аэрокосмические, медицинские, банковские, страховые, транспортные и др. В то время и возникли объектно-ориентированный анализ (OOA = object-oriented analysis), техника объектного моделирования (OMT = object-oriented technique), метод Буча (Booch) и Обжектори (Objectory). Примерно в 1998 году появился метод Шлера (Shlaer) и Меллора (Mellor), который хотя и не являлся полностью объектно-ориентированным, но сыграл в то время важную роль в продвижении объектно-ориентированной парадигмы в проектировании.

OOA

OOA был разработан Кодом (Coad) и Иордоном (Yourdon).
Метод разделяет разработку на три слоя. В первом слое осуществляется определение объектов, поэтому слой и называется объектным. Тут пользователи могут продемонстрировать свое понимание проблемной области, определяя соответствующие объекты из предметной области.
Второй слой называется слоем атрибутов. На этом этапе определяются атрибуты (элементы данных), связанные с объектами из предметной области.
Третий слой – сервисный слой, определяющий сервисы (или операции), предоставляемые каждым объектом.

В сущности, OOA помогает системным инженерам определить требования к системе, вместо того чтобы определять структуру программного обеспечения или его реализацию. В то же время, этот метод может помочь и в описании уже существующей системы (в том числе и ПО), ее операций и того, как другие системы должны взаимодействовать с ней.

OMT

Метод OMT был разработан Румбахом (Rumbaugh).
Основная задача метода это создание таких наборов объектных моделей, детально описывающих архитектуру системы, пока финальный вариант модели не станет пригодным для реализации.
Подход предполагает наличие трех фаз. На фазе анализа создаются модели предметной области. В течение этой фазы создаются модели трех типов – объектные модели, динамические модели и функциональные модели. В первую очередь разрабатываются объектные модели. Для объектных моделей применяется нотация аналогичная OOA, основанная на концепции моделировании сущностей и связей (ER = Entity-Relationship modeling). Объектные модели описывают объекты, классы объектов и связи между объектами. Динамические модели описывают поведение системы с помощью расширения диаграмм состояний Харела (Harel). Функциональные модели описывают функции системы с помощью диаграмм потоков данных (DFD).

Каждая из моделей многократно уточняется и дорабатывается. На фазе проектирования (design) особое внимание уделяется структуре моделей, а на фазе разработки больше внимания уделяется языку, на котором планируется разработка. Таким образом, OMT покрывает не только процесс получение требований к системе, но также и процесс проектирования архитектуры решения.

Booch

Метод Буча (Booch, 1994) был одним из первых предложенных объектно-ориентированных методов. Несмотря на то, что метод рассматривает фазу анализа, его основная сила появляется при проектировании объектно-ориентированных систем. Метод предусматривает постепенное и многократное уточнение архитектуры системы (incremental and iterative approach). Метод также вводит понятия логического и физического представлений системы.

Метод предусматривает фазу анализа проблемной области для выявления классов и объектов, а также связей между ними. Метод использует графическую нотацию для

проектирования. Нотация имеет расширения для реализации классов и объектов, а также сервисов предоставляемых ими. Другой важной особенностью применяемой нотации является наличие диаграмм переходов состояний (state transition diagrams) и временных диаграмм (timing diagrams).

Objectory

Метод Обжектори был предложен Якобсеном (Jacobsen).
Большинством своих идей метод похож на другие объектно-ориентированные методы. Фундаментальным отличием метода, как уже упоминалось выше, является использование сценариев или *прецедентов* (scenarios или use cases). Таким образом, функциональность системы описывается с помощью набора прецедентов для системы – моделью прецедентов.

В дальнейшем, модель прецедентов используется для получения модели объектов предметной области. Анализ полученной модели объектов предметной области позволяет классифицировать объекты предметной области трех типов: интерфейсные объекты, объекты сущностей и управляющие объекты. Далее объектная модель преобразуется в архитектурную модель, содержащую модули разрабатываемой системы.

UML

Универсальный язык моделирования UML (OMG, 2003) создавался как попытка объединить вместе объектно-ориентированные подходы, получившие наибольшую поддержку и признание, а именно: Буч, OMT и Обжектори. В середине 90х годов XX века Буч, Румбах и Якобсен стали вместе работать в компании Rational, начав разрабатывать единый, общий и ныне широко применяемый язык моделирования. В то время максимальное внимание уделялось разработке нотации для графического моделирования, нежели чем разработке метода или процесса.

С момента своего появления UML неоднократно подвергался доработке и изменениям. Было выпущено несколько версий UML. Так, в 1997 году версия UML 1.0 стала международным стандартом, после того как была признана OMG-группой (Object Management Group), а в 1999 была анонсированная уже версия 1.3. В этой книге рассматривается уже версия UML 2.0, выпущенная в 2003 году.
В последующих разделах язык UML описывается более подробно.

3.3.3 Нотация UML

UML предполагает разработку нескольких моделей, совокупность которых описывает разрабатываемую систему. Каждая модель относится к соответствующей фазе и имеет собственное предназначение. При этом каждая из моделей состоит из одной или нескольких UML-диаграмм, которые можно классифицировать следующим образом:

- структурные (structure) диаграммы;
- диаграммы поведения (behaviour);
- диаграммы взаимодействия (interaction).

На рис. 3.24 представлены все 13 типов диаграмм, существующих в нотации UML 2 и доступных системным инженерам.
На практике многие из этих диаграмм не так часто применяются при разработке. Иногда бывает достаточно использовать лишь небольшой набор диаграмм для моделирования системы. Опыт показывает, что наиболее часто используются диаграммы прецедентов, диаграммы классов и диаграммы последовательностей. Если же требуется моделирование динамического поведения системы, то следует воспользоваться диаграммами действий и диаграммами состояний.

Рис. 3.24 Диаграммы UML.

Следует заметить, что цель данного раздела - это не подробное описание UML2, - для этого существуют другие источники, - а демонстрация того, как модели могут использоваться при разработке требований.

Давайте вернемся к примеру с банковской системы, рассмотренной ранее этой главе.
На рис. 3.25 показана UML-диаграмма классов для банковской системы (диаграмма классов является базовой в UML). На диаграмме показан набор классов, используемый для моделирования системы, - классы «Счет», «Владелец», «Текущий Счет» и «Выписанный Чек». Как видно из диаграммы, каждый класс имеет собственное имя, набор атрибутов и

операций, а также связи с одним или более классов (в данном случае *обобщения* и *ассоциации*).

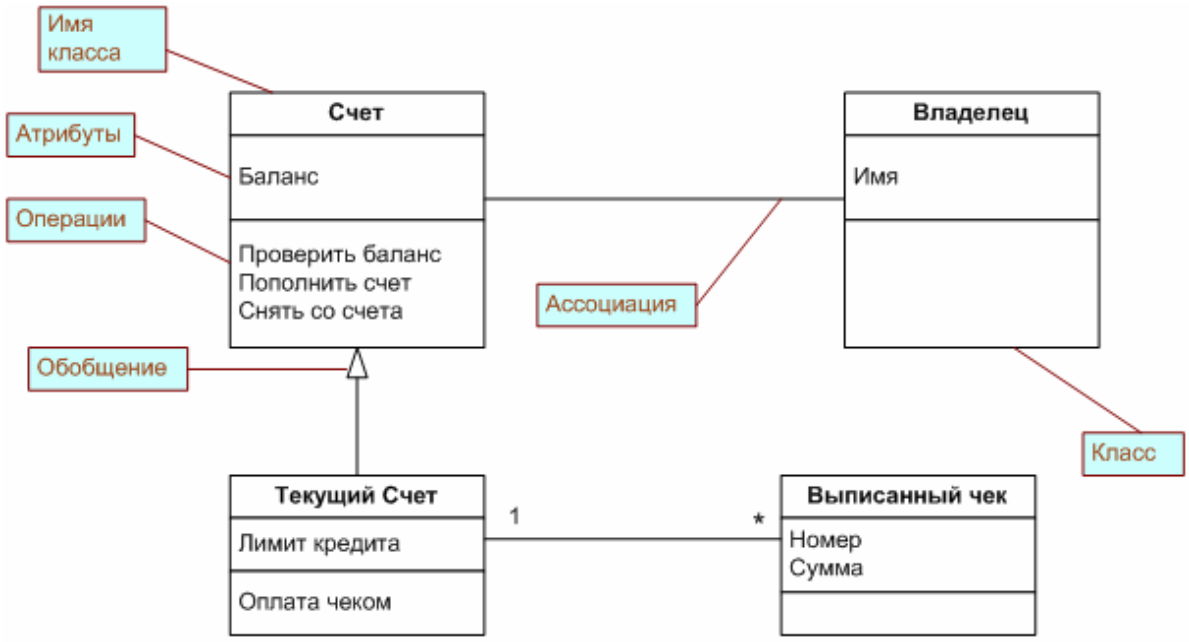


Рис. 3.25 Расширенная диаграмма классов UML для банковской системы.

Моделирование также бывает полезным, когда существуют внешние системы, или устройства, которые разрабатываемая система будет использовать. Эти системы могут быть представлены в виде классов.

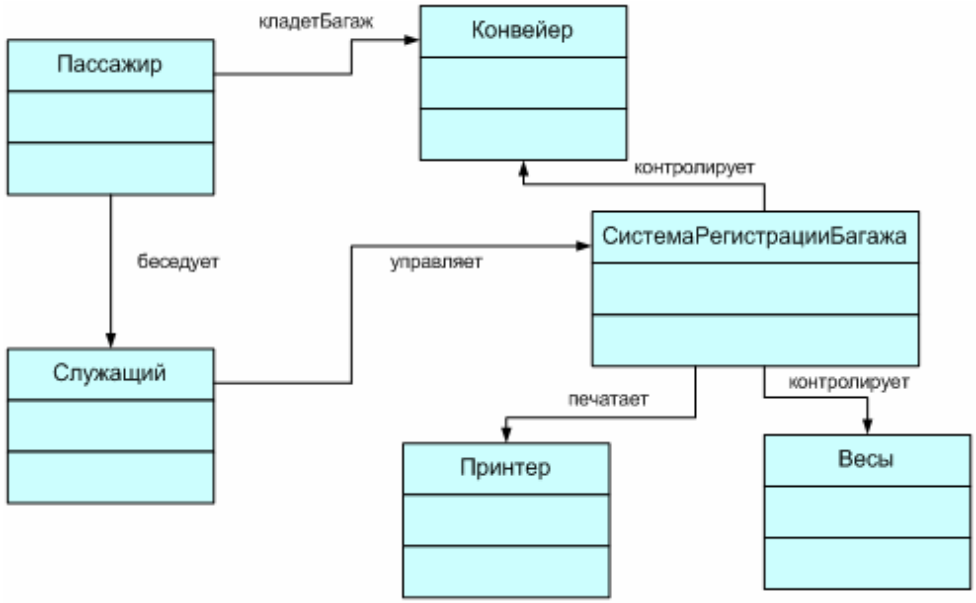


Рис. 3.26 Диаграмма классов для системы транспортировки багажа.

На рис. 3.26 приведен другой пример – диаграмма классов для системы транспортировки багажа.
Диаграмма отражает пользовательские требования, которые явно относятся к проблемной области. Для системы транспортировки багажа были определены следующие классы: «Пассажир», «Служащий» и «Конвейер», а также были выделены две встроенные системы – «СистемаРегистрацииБагажа» и «Весы». Ассоциации между системами и классами помогают более четко определить контекст работы системы.

Переходя к области решений, необходимо задуматься о функционале и поведении системы. Следовательно, диаграмма классов должна быть уточнена и дополнена так, чтобы отобразить все эти атрибуты (классов), которые будут необходимы для последующего моделирования системных требований (см. рис. 3.27).

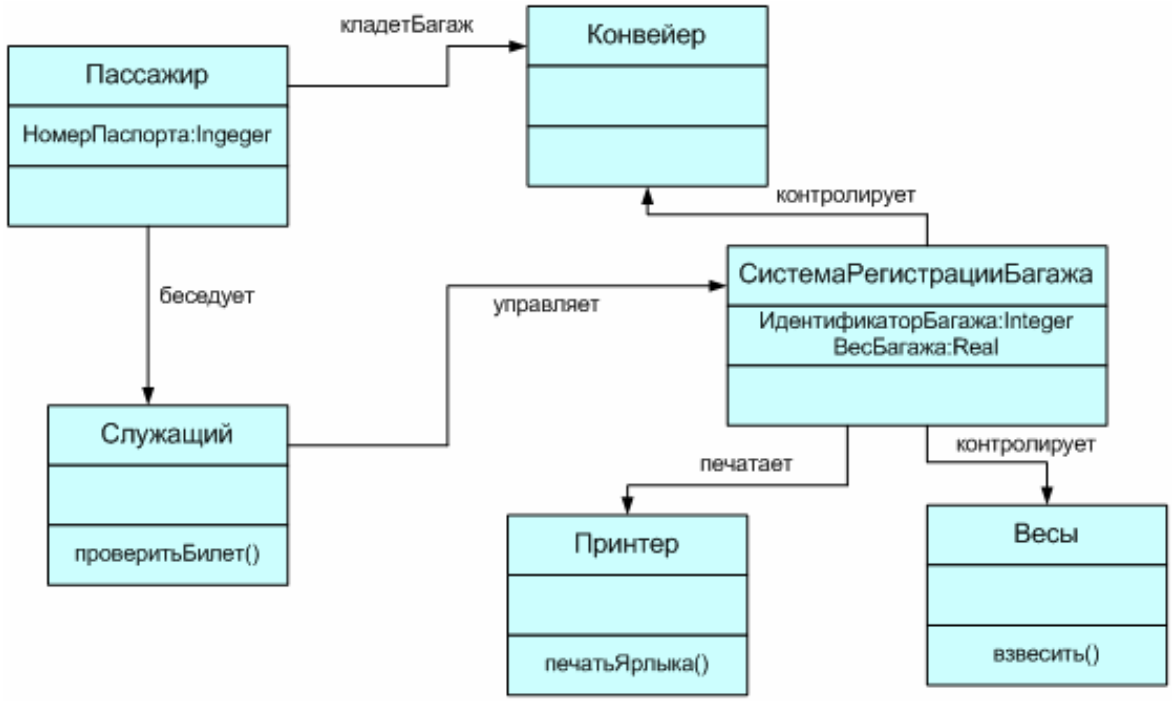


Рис. 3.27 Уточненная диаграмма классов.

Для описания функциональных требований к системе используется моделирование прецедентов. В качестве примера можно рассмотреть две диаграммы прецедентов – одна для системы транспортировки багажа, а другая для системы регистрации багажа.
На рис. 3.28 первая из систем представлена как система верхнего уровня.
На рис. 3.29 изображена диаграмма прецедентов для системы регистрации багажа.
На обеих диаграммах определены относительные границы систем (обозначены прямоугольником), а также различные типы заинтересованных сторон (shareholders), называемые актерами, которые находятся за границами системы. Необходимо отметить, что прецеденты представляют собой высокоуровневые цели заинтересованных сторон. Связь типа «включает» показывает, что этот прецедент является частью другого прецедента (включается), т.е. таким образом показывается иерархическая декомпозиция прецедентов.

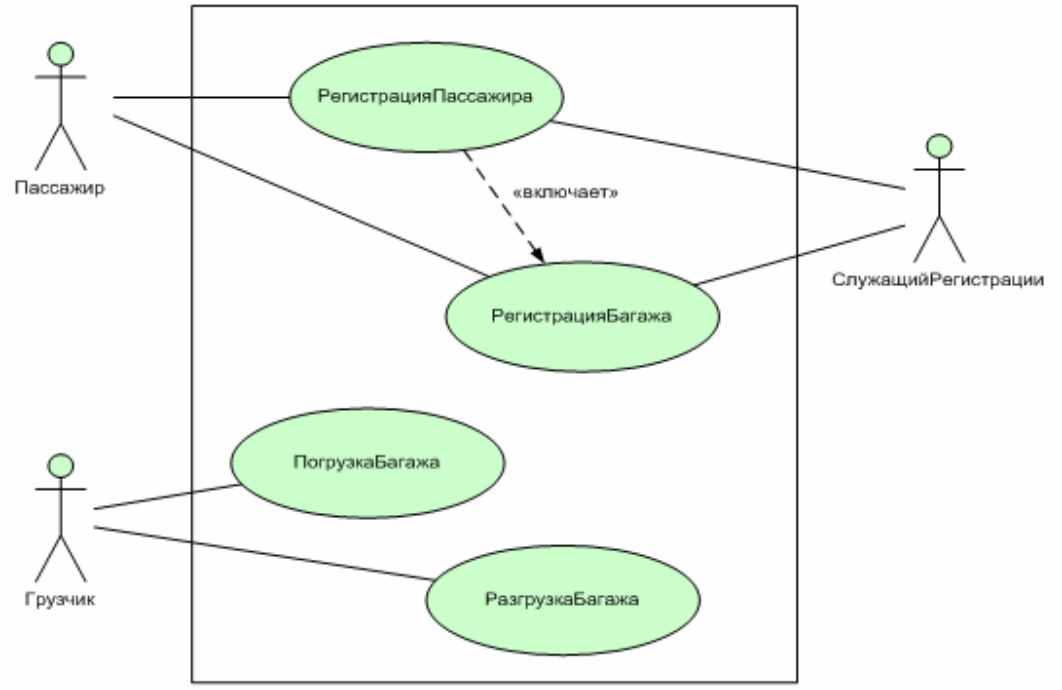


Рис. 3.28 Диаграмма прецедентов для системы транспортировки багажа.

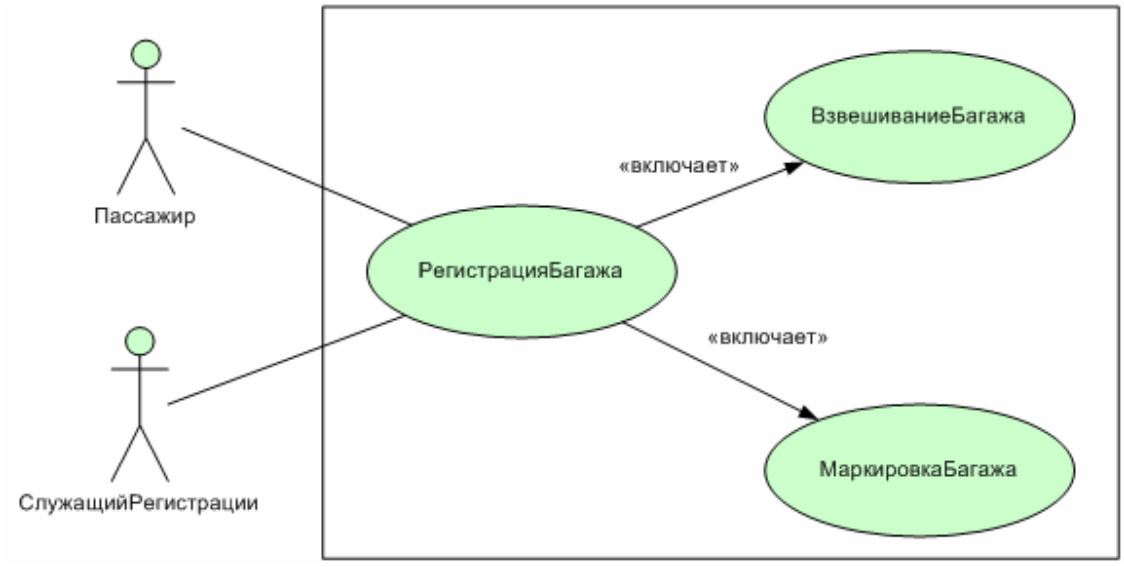


Рис. 3.29 Диаграмма прецедентов для системы регистрации багажа.

UML также дает проектировщикам возможность моделировать функциональность системы и ее поведение. Диаграмма последовательности показывает взаимодействие и совместную работу объектов, и таким образом позволяет моделировать сложное поведение системы с

помощью сообщений, которыми обмениваются объекты друг с другом в процессе взаимодействия. На рис. 3.30 приведен пример диаграммы последовательности. В верхней части диаграммы прямоугольниками изображены объекты, каждый имеет вертикальную линию, обозначающую время. Сообщения изображаются в порядке их появления с помощью стрелок между линиями времени объектов. Одна диаграмма последовательности может ссылаться на другую с помощью специального символа – прямоугольника с названием другой диаграммы последовательности и пометкой ref (“reference”). Это обозначает, что часть взаимодействия между объектами в этом промежутке времени изображена на другой диаграмме. В нашем случае - это «ВзвешиваниеБагажа» и «МаркировкаБагажа».

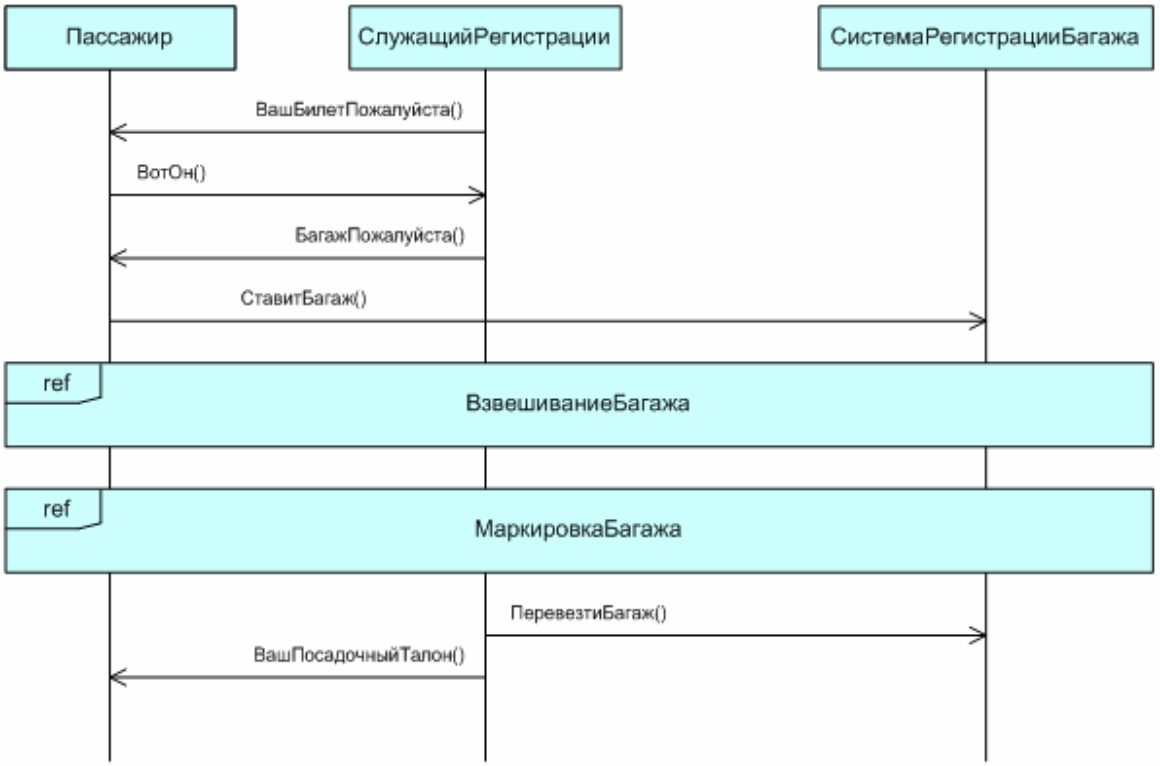


Рис. 3.30 Пример диаграммы последовательности.

С помощь вынесения описания части взаимодействия в отдельные диаграммы достигается возможность повторного их использования в модели.

3.3.4 Формальные методы

Формальные методы обеспечивают более строгое представление, основанное на математике, и, таким образом, могут быть использованы для математического обоснования полноты спецификаций и правильности их исполнения. Применение формальных методов дает возможность строгой проверки позволяющей избавиться от ошибок определенных видов. Использование таких методов необходимо для определенных классов систем,

например для атомной энергетики, разработки оружия, систем управления воздушным транспортом.

Z (Spivey, 1989), VDM (Jones, 1896), LOTOS (Bjorner, 1987) и B (Abrial, 1996) являются наиболее распространенными формальными методами для формального описания функциональности. LOTOS (Language of Temporal Ordering Specification – язык для определения временной последовательности), VDM (Vienna Definition Language – язык венского определения) и Z стандартизированы ISO. Языки B и LOTOS исполняемые, B также может быть переведен в программный код.

Формальные методы применяются для критических систем, там, где потенциальные человеческие или финансовые потери могут быть очень значительны и стоимость применение точных математических методов, таким образом, вполне оправдана.

Формальные методы постепенно приобретают большую значимость. Если бы рамки применения формальных методов были шире, чтобы охватить большее количество аспектов системного проектирования, они бы стали более полезны и, следовательно, популярны.

Z–метод для формального моделирования

Z - это нотация для моделирования основанная на логике предикатов первого порядка и теории множеств. Нотация позволяет представлять данные в виде множеств, отображений, кортежей, связей, последовательностей и декартовых произведений. Также существуют функции и символы операций для обработки данных этих типов.

Z-спецификации представляют собой маленькие, легко читающиеся элементы, называемые «схемами». Схема состоит из части объявления переменных и части предиката. В разделе объявления переменных содержится список объявленных переменных, в разделе предиката всегда содержится только один предикат. Наименование схемы вводит синтаксическую эквивалентность между именем и схемой. На рис. 3.31 изображена Z-схема.

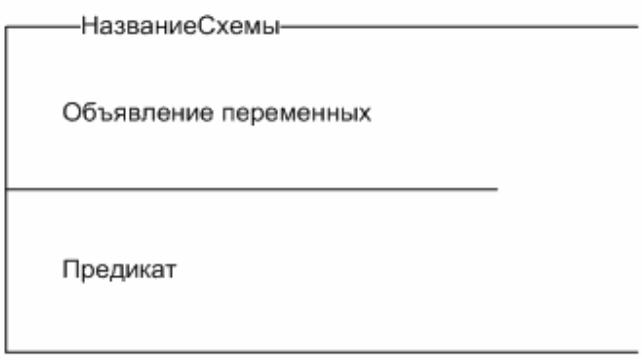


Рис. 3.31 Z-схема.

Спецификации Z представляют собой набор схем, где схемы представляют собой описания сущностей и связей между ними. Схемы, таким образом, обеспечивают среду для разработки спецификаций системы и их постепенного развития и уточнения.

На рис. 3.32 показана Z-схема для операции «**выдача**» (книги) в библиотеке, где общее поведение библиотеки как системы могло бы быть представлено с помощью схемы под названием «**библиотека**». Условное обозначение **Δ Библиотека** называется «дельта схема» и обозначает, что схема «выдача» вызывает изменение состояния в Библиотеке.

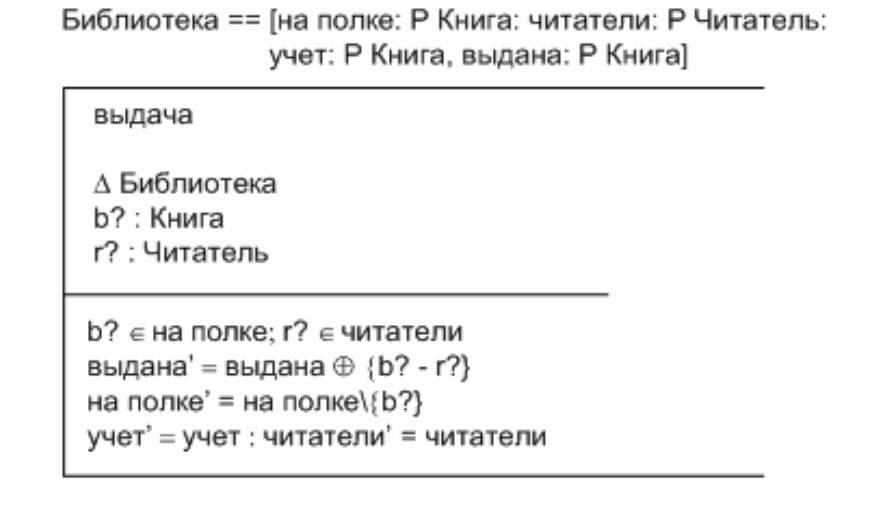


Рис. 3.32 Пример схемы.

Схема на рисунке 3.32 разделяет вход и выход, и состояние до и после. Эти операции обозначаются следующим образом:

- **?** - обозначает входящую переменную для операции;
- **!** - обозначает переменную на выходе операции.

Состояние после операции обозначается штрихом, например, **учет′**, для отделения его от состояния до операции.

3.4 Заключение

В главе были рассмотрены вопросы моделирования, в основном применительно к области решений. Были рассмотрены различные техники и методы, начиная с тех, которые уже прошли проверку временем, заканчивая теми, которые появились совсем недавно. Тем не менее, все рассмотренные методы и техники находят очень широкое практическое применение.
Материал представленный в данной главе обеспечивает основу для рассмотрения моделирования пользовательских и системных требованиях в последующих главах книги.

4 Написание и анализ требований

Писать просто и ясно так же трудно, как быть искренним и добрым.
Сомерсет Моэм, (William Somerset Maugham)
писатель, 1874–1965

4.1 Введение

Разработка требований это технический процесс. Поэтому написание требований в корне отличается от написания других текстов, т.е. этот процесс совершенно непохож на написание романов или книг. Написание требований также непохоже и на написание обычной технической документации, например, такой как инструкция по эксплуатации или руководство для пользователей.

Целью данной главы является ознакомление читателя с такими характерными аспектами написания требований, которые являются общими для любого уровня разработки. Как бы не выглядел этот ваш основной процесс разработки, каким бы изменениям он не подвергался, определенная техника и принципы формулирования и структуризации требований остаются неизменными.

Необходимо помнить, что в процессе написания требований два очень важных аспекта должны быть аккуратно сбалансированы:

- документ с требованиями должны быть удобным для чтения;
- наборы требований должны быть удобными для работы с ними.

Под первым подразумевается, что документ с требованиями должен быть структурирован таким образом, чтобы пользователю было легко понять формулировку каждого индивидуального требования в контексте всего документа.
Под вторым аспектом подразумевается качество каждого отдельного требования, каким языком оно написано, насколько оно четко и точно отражает суть, насколько требование может быть представлено в виде некоего элемента, с которым удобно устанавливать связи от других требований.

Опытные специалисты, работающие в области написания требований, хорошо понимают, что как такового текстового редактора явно недостаточно для управления набором требований, создания структуры, классификации, определения свойств и установления связей между требованиями. В качестве иллюстрации неудобства использования текстового редактора можно привести пример, когда нумерация требований осуществляется с помощью номеров параграфов. Попытка вставить в середину документа новое требование приведет к тому, что все последующие требования автоматически изменят свою нумерацию.

Аналогичным образом, те, кто пытается работать с требованиями с помощью баз данных, вскоре приходят к выводу, что наличие многочисленных таблиц, набитых индивидуальными требованиями, все равно практически не позволяет управлять ими как

единой и цельной структурой. Т.е. несмотря на то, что с помощью базы данных легко нумеровать, классифицировать и сортировать требования, жизненно важный смысл всего документа утрачивается, поскольку после таких операций каждое конкретное требование теряет свой смысл, будучи вырванным из общей структуры документа.

Таким образом, оба аспекта – целостность документа и качество отдельного требования должны постоянно находиться под пристальным вниманием.

При этом следует отметить, что написание требований и их анализ (рецензирование) должны происходить параллельно. По той простой причине, что критерии, используемые для написания хороших требований, те же самые, что и для рецензирования (анализа) последних.

Именно поэтому написание и анализ (рецензирование) требований рассматриваются нами вместе в одной главе.

4.2 Требования для требований

Прежде чем приступать к обсуждению того, как лучше формулировать требования и составлять документы, необходимо вначале проанализировать общие цели и задачи разработки требований. Это поможет понять суть предлагаемых принципов, которые излагаются в данной главе.

Отправной точкой при разработке требований является определение заинтересованных сторон (stakeholders), как это показано в таблице 4.1.

Таблица 4.1 Заинтересованные стороны (stakeholders) для требований

Заинтересованная сторона	Роль
Автор	Создает требования и оформляет изменения
Издатель	Выпускает и архивирует документ требований
Цензор	Рецензирует требования и предлагает изменения
Конструктор, системный аналитик, разработчик	Анализирует требования и обсуждает изменения

В таблице 4.2 представлены те возможности (полезные свойства), предоставляемые требованиями, которые необходимы различным заинтересованным сторонам.
В таблице перечислены все основные действия, которые могут быть совершены с требованиями, включая определение, классификацию, оценку, контроль статуса, контроль связей, рассмотрение в контексте всего документа, анализ и рецензирование. То, насколько хорошо написаны требования, и насколько хорошо организован набор требований, существенным образом влияет на удобство работы с этим набором требований.

Таблица 4.2 Возможности, которые должны предоставлять требования

Возможность
• Возможность однозначно идентифицировать каждое положение требования • Возможность классифицировать каждое положение требования различными способами, например: - по важности - по типу (например, функциональность, производительность, ограничение, безопасность) - по срочности (дата реализации) • Возможность отслеживать статус каждого требования с разных точек зрения, таких как: - статус рецензирования (анализа) - статус удовлетворения - статус проверки • Возможность оценивать требование с разных сторон, таких как: - информации о производительности - стратегия проверки - критерии тестирования - рациональность - комментарии • Возможность анализировать каждое требование в контексте целого документа, т.е. в окружении других требований • Возможность нахождения в документе определенного требования по контексту, классификации или другим признакам • Возможность установления связей между требованиями и легкого перехода по этим связям между требованиями

4.3 Разработка структуры требований

Документация с требованиями может занимать очень большие объемы. Так, например, полный набор требований для авианосца в печатном виде может занимать несколько полностью набитых шкафов. Для того, чтобы доставить такой объем документов от поставщика к заказчику, потребуется даже не один грузовик. В такой ситуации ясная хорошо продуманная структура требований и документов имеет очень большое значение для облегчения управления и реализации всего комплекса требований.

Создание хорошей структуры требований может помочь:

- *минимизировать* общее количество требований;
- *лучше осмыслить* большой объем информации;
- *отыскать* наборы требований, относящихся к определенной теме;
- *выявить* пробелы и повторения;
- *исключить* конфликт (противоречия) между требованиями;
- *управлять* этапами реализации (например, вначале отложенных требований);
- *отклонить* малоинформативные требования;
- *оценить* требования (напр., с точки зрения стоимости или времени реализации);
- *повторно использовать* требования в последующих проектах.

Обычно документ, содержащий требования, состоит из множества секций и подсекций, т.е. уже имеет иерархическую структуру. Такая иерархия секций достаточно удобна для начальной классификации требований, поскольку позволяет использовать структуру заголовков для распределения требований по категориям. При таком подходе та позиция, которую занимает требование в общей структуре документа, является его *первичной* классификацией. (*Вторичная* классификация требования может быть организована посредством связей с требованиями других секций или же с помощью атрибутов).

В главе 3 описывается, как часто в системном анализе используются иерархические структуры при моделировании. В качестве иллюстрации можно привести следующие примеры:

- декомпозиция целей и возможностей в пользовательских сценариях;
- функциональная декомпозиция в диаграммах потоков данных;
- декомпозиция состояний системы в диаграммах состояний.

В случае, когда требования формируются на основе анализа моделей, структура этих моделей может использоваться как часть структуры документа.

Помимо формулировки самих требований документ, в котором собраны требования, может также содержать большое количество разного рода технического и нетехнического текста, помогающего лучшему пониманию сути требований.
К такого рода текстам может относиться:

- *Сопроводительная информация*, которая помогает правильно позиционировать требование в контексте документа;
- Описание *внешнего окружения* системы или, как это часто называют, «знания о предметной области»;
- *Определение границ* требований (что включать, а что нет – т.е. границы проекта);
- *Определение терминологии*, используемой в документе;
- *Описательный текст*, который служит для связи разделов документа между собой;
- *Характеристика заинтересованных сторон*;
- *Краткое описание моделей*, используемых для получения требований;
- *Ссылки* на другие документы.

4.4 Понятие о ключевых требованиях

Многие организации начинают применять концепцию «ключевых требований» практически уже на уровне пользовательских требований.
Такие требования, часто называемые ключевыми пользовательскими требованиями KURs (key user requirements) или ключевыми показателями производительности KPIs (key performance indicators), являются небольшой «выжимкой» из общих требований, описывающих суть системы (ее основные функций).

При выборе ключевых требований нужно руководствоваться той же философией, что и герои романа Джерома К. Джерома «Трое в лодке, не считая собаки», которые, собираясь в путешествие, пришли к следующему умозаключению:

Несомненно, Темза в своем верхнем течении недостаточно судоходна, и по ней не сможет подняться судно, которое вместит все, что мы сочли необходимым взять с собой.

...

Джордж сказал: - «… Нужно думать не о том, что нам может пригодиться, а только о том, без чего мы никак не сможем обойтись»

Каждое ключевое требование должно подразумевать отрицательный ответ на вопрос:

> *Если решение не предполагает <эту> возможность (функцию, опцию, т.д.), стану ли я в этом случае его приобретать?*

или то же, но на системном уровне:

> *Если система не обеспечивает <этого>, будет ли система все еще нужна мне?*

В этом случае ключевыми требованиями смогут называться только те, которые абсолютно необходимы. (Разумеется, любые требования могут обсуждаться и корректироваться, - в том числе и ключевые, - однако надо помнить, что обсуждение ключевых требований всегда должно проходить с большим вниманием и осторожностью).

Там, где это уместно, каждому ключевому требованию KUR должен быть поставлен в соответствие показатель производительности. В этом случае ключевые требования могут быть использованы как ключевые показатели производительности KPIs для оценки эффективности предлагаемых альтернативных решений.

4.5 Использование атрибутов

Из содержания предыдущих глав, а также из возможностей требований, перечисленных в таблице 4.2, становится ясным, что наличие простого текстового описания явно недостаточно, чтобы полно и однозначно определить требование. Для полного определения требования необходима также и другая информация, которая ему присуща, - признаки классификации, статуса и др.

Чтобы не перегружать формулировку требования излишними деталями, специалисты рекомендуют выносить всю дополнительную информацию в атрибуты, жестко привязанные к требованию.

[SH234] Система управления скорой помощью должна быть способна принимать до 100 ста вызовов одновременно

Автор:	R. Thomas
Приоритет:	Обязательное
Релиз:	1
Статус рецензирования:	Одобрено
Возможность проверки:	Да
Способ проверки:	Симуляция, затем системные тесты

Рис 4.1 Атрибуты требования.

Используя содержимое атрибутов, намного легче обрабатывать требования, осуществлять поиск и выборку, сортировку и т.п. Атрибуты могут быть использованы для поддержания большинства возможностей требований, перечисленных в таблице 4.2, делая сам процесс разработки требований более наглядным, управляемым и удобным.
На рис. 4.1 показан пример требования [SH234] с присущими ему атрибутами.

Использование того или иного конкретного набора атрибутов зависит от используемого компанией процесса разработки требований, который необходимо поддержать. При этом значения некоторых атрибутов могут заполняться автоматически, например, последовательная нумерация или дата; значения других заполняются пользователями (или со слов пользователей), например, приоритет или номер релиза; суть значения третьих – это флаг, устанавливаемый после аналитической работы с требованиями, например, пригодность для проверки.

В таблице 4.3 в качестве наглядного примера, иллюстрирующего вышесказанное, приведены те категории атрибутов требований, которые использовались английским подразделением INCOSE (The International Council on Systems Engineering), работавшим над одним из проектов.

4.6 Связанность и согласованность требований

При работе с большими наборами требований зачастую достаточно трудно идентифицировать те требования, которые могут противоречить друг другу по смыслу. Согласитесь, что, если не иметь специальных средств для выявления подобных конфликтов, не так-то просто понять, что требование, находящееся через несколько страниц от данного, имеет противоположный смысл. Что же может помощь в этом случае?

Ответ достаточно прост. Необходимо иметь возможность классифицировать, фильтровать и сортировать требования с тем, чтобы иметь возможность получать относительно небольшую выборку требований, относящихся к одной теме, для последующего анализа.
При этом многие требования могут одновременно затрагивать различные аспекты функционирования системы. Например, требование, относящееся в основном к вопросу производительности двигателя, может затрагивать и вопросы безопасности. В этом случае данное требование должно рассматриваться как в контексте производительности двигателя, так и в контексте безопасности.

Для поддержания такой возможности требования должны иметь первичную и вторичную классификацию (как обсуждалось в разделе 4.3). Обычно каждое требование имеет *единственную первичную* классификацию (например, его месторасположение в контексте документа) и *множественное количество вторичных* классифицирующих свойств, использующих возможности атрибутов и связей.

Эта техника существенным образом помогает при анализе и рецензировании требований, позволяя находить все связанные между собой по смыслу требования с помощью фильтрации и сортировки по ключевым словам и используя признаки основной и дополнительных классификаций.
Например, вначале, чтобы сузить поле возможного поиска, вы строите выборку всех требований, относящихся к безопасности. А затем уже, среди отобранных, вы анализируете схожие требования на предмет наличия конфликтов между ними.

Таблица 4.3 Категории атрибутов

Категория	Примеры значений
Идентификация	
• Идентификатор	Уникальный номер требования (ID)
• Название	Уникальное краткое название, характеризующее требование
Внутренние характеристики	
• Основной тип	Функциональность, производительность, качество, окружение, интерфейс, ограничение, не требование
• Качественный подтип	Доступность, гибкость, целостность, ремонтопригодность, портативность, легкость поддержки, легкость использования, квалификация
• Тип продукта/процесса	Продукт, процесс, данные, сервис
• Количественный/качественный тип	Количественный, качественный
• Фаза жизненного цикла	Предварительная концепция, окончательная концепция, разработка, производство, интеграция/тестирование, внедрение/поставка/установка, функционирование, поддержка, удаление/демонтаж
Приоритет и важность	
• Приоритет	Ключевое, необходимое, дополнительное, желательное (Key, mandatory, optional, desirable) или Обязательное, рекомендуется, возможное, желательно (Must, Should, Could, Wish)
• Важность	Шкала от 1 до 10
Источник и владелец	
• Способ получения	Назначение, декомпозиция
• Источник	Название документа или имя заинтересованного лица
• Владелец	Имя заинтересованного лица
• Согласовано	Имя человека
Контекст	
• Набор требований/документ	(наилучшим образом управляется с помощью правильного расположения требования в структуре требований)
• Объект	
• Границы (рамки)	
Проверка и утверждение (verification & Validation, или V&V)	
• V&V метод	Анализ, инспекция, системный тест, модульный тест
• V&V стадия	(см. Фаза жизненного цикла)
• V&V статус	В очереди, проверено, отклонено, не завершено
• Критерий успешности проверки	Зависит от выбранной декомпозиции
• Критерий утверждения	Зависит от выбранного V&V метода
Поддержка процесса	
• Статус согласования	Предложено, на согласовании, согласовано
• Статус проверки	Проверено, не проверено, подозрительно
• Статус удовлетворения	Неудовлетворенно, удовлетворено, подозрительно
• Статус рецензирования	Ожидает анализа, принято, отклонено
Уточнение	
• Необходимость	Описание того, почему возникла необходимость в данном требовании
• Комментарии	Текстовое уточнение требования
• Вопросы	Вопросы, которые нужны для уточнения требования
• Ответы	Ответы, полученные при уточнении
Прочее	
• Зрелость (стабильность)	Количество изменений/время
• Уровень риска	Высокий, средний, низкий
• Оценочная стоимость	
• Фактическая стоимость	
• Релиз продукта	Версия продукта, в которой реализовано данное требование

4.7 Важность требования

Некоторые требования можно отнести к категории «не обсуждаемых».
Т.е. их следует не обсуждать, а именно выполнять, потому как, если конечный продукт не удовлетворяет таким требованиям, то он просто не будет использовать.
Соответственно, другие требования могут обсуждаться и корректироваться.
Так, например, если в соответствии с требованиями система управления работой скорой помощи должна обеспечивать одновременную работу как минимум 100 пользователей, а готовое решение поддерживает только 99 пользователей, работающих одновременно, то такое решение, вероятнее всего, будет все-таки признано удовлетворительным и полезным заказчику и пользователям.

Оценить степень важности (удовлетворенности) требования может быть само по себе трудной задачей. Возможно, что для предыдущего примера достижение показателя в 75 одновременно работающих пользователей будет приемлемой величиной, а вот любая величина ниже 50 будет уже категорически неприемлема, но показатель в 200 пользователей скорей всего будет очень хорошим результатом, который для заказчика будет даже более ценен, чем 100.

Одним из подходов, облегчающих решение этой проблемы, является определение нескольких значений производительности для одного показателя. Ниже приведен пример для трех значений:

- **О** (обязательный): Обязательный верхний (или нижний) предел значения величины;
- **Ж** (желаемый): Желаемое значение;
- **Н** (наилучший): Наилучшее значение.

Каждое из этих значений может храниться в собственном атрибуте требования или же они могут быть описаны непосредственно в тексте требования, например, в такой форме: «Система должна поддерживать функционирование [О:50, Ж:100, Н:200] одновременно работающих пользователей».

Другой подход заключается в том, чтобы графически (с помощью функции) отобразить значение важности требования в зависимости от показателя производительности. В этом случае важность требования обычно находится в пределах от 1 до 100 единиц.
На рис. 4.2 показаны четыре примера, отображающие различную форму функции значения важности требования.
Функция (a) демонстрирует случай, когда заказчику желательно, чтобы число пользователей, работающих одновременно, стремилось к максимуму, но при этом определен и минимально допустимый показатель – 50 пользователей.
Функция (b) иллюстрирует бинарный случай - либо определенная производительность (в 100 единиц) достигается, либо нет. При этом даже 200 одновременно работающих пользователей не приносят никакой дополнительной пользы заказчику.
Функция (c) отражает случай, когда значение показателя должно быть минимизировано (например, вес устройства), но при этом определен и максимально допустимый вес – 50 кг.
Функция (d) - это когда значение показателя должно быть оптимизировано (например, обороты двигателя).

Использование графических функций является весьма наглядным способом представления важности требования. Один взгляд на функцию дает представление о сути

требования – требуется ли минимизировать, максимизировать, оптимизировать и т.д. показатель.

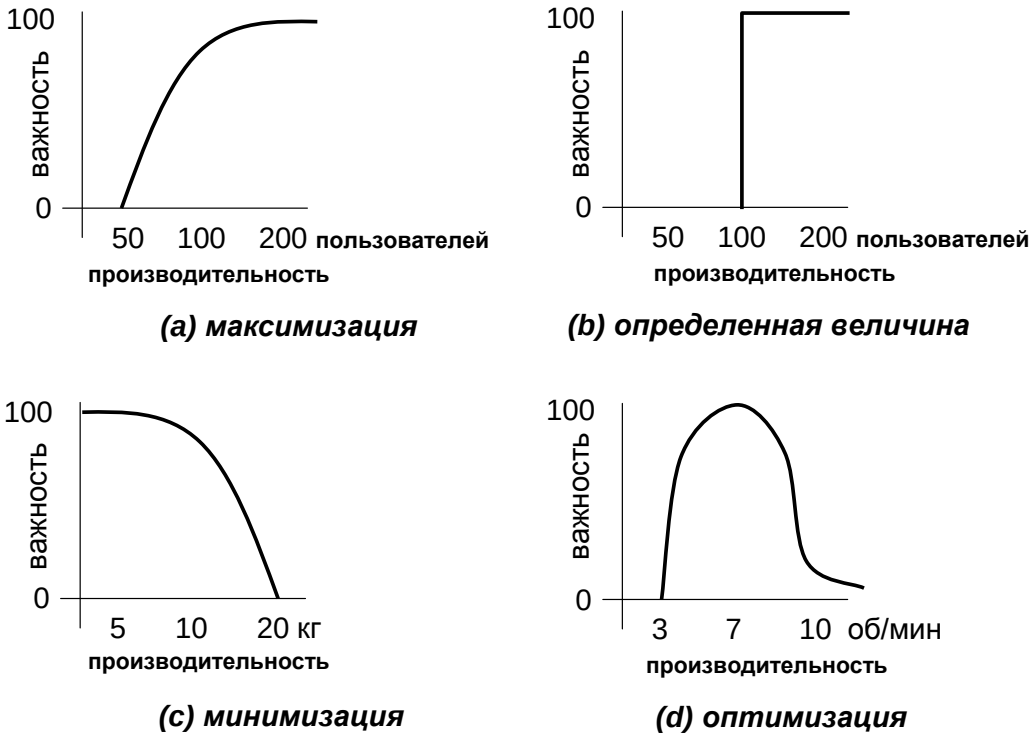


Рис. 4.2 Типичные функции важности.

В качестве дополнительного преимущества такой подход дает инженерам возможность осознать степень их свободы при разработке решения, т.е. путем согласования значений показателей производительности для отдельных требований получить в результате наилучшие значения производительности всей системы в целом.
Такой подход весьма часто используется при проведении тендеров для сравнения и оценки однотипных критериев альтернативных предложений[6].

Для отображения значения требования может использоваться также и атрибут, содержащий конкретное значение функции в зависимости от производительности системы.

4.8 Язык требований

Использование четкого и ясного языка (согласованных терминов) при написании требований позволяет существенном образом облегчить последующее понимание требований и их классификацию.
Простым примером здесь может служить использование в тексте слова «должен» (должна, должно), как ключевого слова, обозначающего наличие требования. Некоторые подходы

[6] В частности, этот принцип использует Telelogic Assessment Management, базирующийся на Doors.

даже предписывают использование отличающихся ключевых слов для характеристики приоритета требований, например – «должно» (*must*), «рекомендуется» (*should*) и «возможно» (*may*).

Следует заметить, что язык, используемый для написания требований, в значительной степени зависит от «уровня» документа с требованиями, т.е. существует принципиальное отличие между пользовательскими требованиями, которые относятся к проблемной области, и системными требованиями, которые относятся к области решений (см. раздел 1.7).

Как подчеркивалось в главе 5, пользовательские требования в основном описывают возможности (услуги), необходимые пользователям (т.е. потребности пользователей), или ограничения, связанные с этими возможностями или потребностями. И в этом случае требование, описывающее такую потребность, должно описывать только одну потребность, необходимую или для одного пользователя, или группы из нескольких однотипных пользователей.

При этом в тексте требования должен указываться типа пользователя.

Типичное требование, описывающее возможность (потребность), выглядит следующим образом:

<Тип пользователя> должен иметь возможность <описание возможности>

Если существуют определенные требования к производительности или ограничения, связанные только с одним конкретным требованием, то, в этом случае, текст требования может быть дополнен и выглядеть уже следующим образом:

<Тип пользователя> должен иметь возможность <описание возможности>
с <показатель производительности> от <момент отсчета>,
находясь в <условия эксплуатации>

Так, например, выше сформулированное общее требование в частном случае может выглядеть так (содержать условия производительности и ограничения):

Оператор* должен иметь возможность *произвести выстрел
в *течение 3 секунд* с *момента обнаружения цели радаром*,
находясь в *сложных морских условиях*

Гораздо реже встречается ситуация, когда атрибут с одним и тем же условием производительности связан с несколькими разными требованиями. Можно представить, например, ситуацию, когда несколько различных требований характеризуются одним и тем же временным параметром.

Однако на практике, когда существует иерархия требований и требования более низкого уровня являются детализацией требования более высокого уровня, это зачастую означает, что требуемое значение атрибута производительности фактически связано с требованием более высокого уровня, а, следовательно, все требования, являющиеся его детализацией, просто наследуют это же значение атрибута.

Часто можно видеть, что ограничения описываются не в тесте самого требования, описывающего возможности (потребности), а отдельно. Это может быть связано с тем, что такое ограничение либо относится целиком ко всей системе и нет смысла многократно

повторять его в каждом требовании, либо, наоборот, эти ограничения касаются как разных аспектов системы и их необходимо выделять для последующего контроля.

Обычно ограничения в пользовательских требованиях упоминаются или как минимально приемлемые параметры производительности, заявляемые заказчиком, или являются следствием необходимости взаимодействия создаваемой системы с окружением (сюда же, что характерно, относятся правовые и социальные системы).

Требование типа ограничение обычно выражается в следующей форме:

> **<Тип пользователя> не должен попадать под действие <соответствующе законодательство>**

Пример из реальной жизни:

> ***Водитель скорой помощи* не должен подпадать под действие *законодательства, предусматривающего ответственность за нарушение правил дорожного движения.***

Поскольку ограничения относятся к области решений, то язык системных требований отличен от языка пользовательских требований. При формулировке системных требований основной акцент делается на описание системные функции и построение ограничений. Конкретная формулировка требования зависит также от типа ограничения или показателя производительности, которые связаны с этим требованием.

Приведем общий пример описания функции (системное требование), которое содержит требуемое значение показателя производительности (в данном случае нагрузки):

> **<Система> должна <выполняемая функция>**
> **не менее чем <количество> <объект>**
> **функционируя в <условия эксплуатации>.**

Или в реалиях:

> ***Телекоммуникационная система* должна *обеспечивать телефонную связь***
> **не менее чем с *10 абонентами*,**
> **функционируя в *условиях отсутствия внешнего источника электропитания***

Приведем другой пример, описывающий периодическое ограничение:

> **<Система> должна <выполняемая функция> <объект>**
> **каждые <показатель производительности> <единица измерения>**

На языке инструкции это выглядит:

> ***Кофе-машина* должна *производить горячий напиток***
> **каждые *10 секунд***

Мы продолжим обсуждение этой темы в последующих разделах.

4.9 Шаблоны требований

В предыдущем разделе был проиллюстрирован язык написания требований с использованием шаблонов. В данном разделе мы продолжим обсуждение этой темы применительно к сбору и формулировке требований типа ограничения.

Использование шаблонов, как это показано на примерах, приведенных в предыдущем разделе, является хорошим способом стандартизации языка, применяемого для разработки требований. При этом для того, чтобы иметь возможность писать стандартным способом требования различных типов, необходимо просто собрать определенный набор таких шаблонов. В процессе применения этого набора на практике, он может уточняться, расширяться и корректироваться с тем, чтобы получить более полный набор шаблонов, который в дальнейшем может даже использоваться и в других проектах.

Таким образом, процесс создания требований с помощью шаблонов может быть разбит на два этапа:

- выбор наиболее подходящего шаблона из общего набора шаблонов;
- подстановка конкретных данных для заполнения пустующих полей в шаблоне.

Такой подход дает возможность разделить шаблон и данные, подставляемые в него. Тогда любое требование будет просто содержать ссылку на шаблон, а данные могут фиксироваться отдельно - как атрибуты этого требования.
На рис. 4.3 приведен пример, иллюстрирующий вышесказанное.

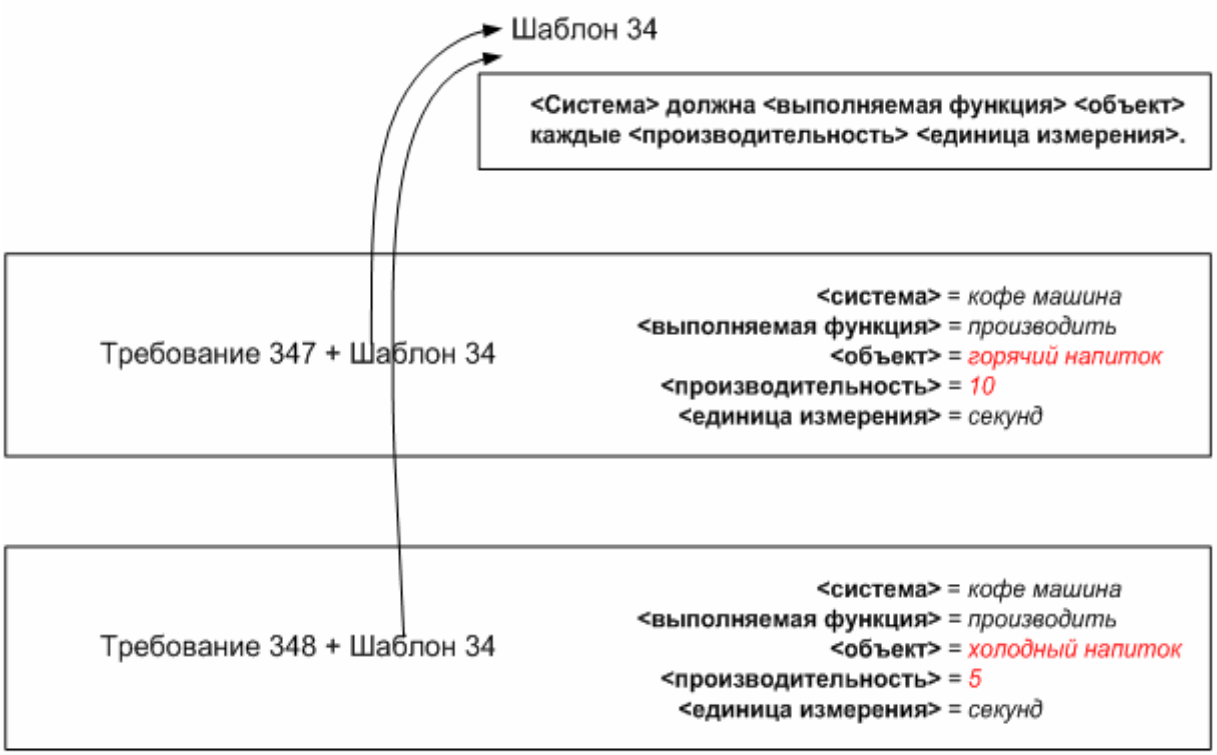


Рис. 4.3 Глобальные шаблоны.

Такой «шаблонный» подход дает возможность сгенерировать текстовое представление требования в любой нужный момент. Использование выделенных шаблонов имеет следующие преимущества[7]:

- *Возможность глобального изменения стиля:* для изменения формулировки определенных требований необходимо внести изменение только в один или несколько конкретных шаблонов, которые задействованы в этой схеме.
- *Возможность более легкой обработки информации:* например, выделение всех полей, относящихся к <условиям эксплуатации>, в отдельный атрибут требований позволяет более удобно фильтровать и сортировать требования, исходя из конкретных признаков условия эксплуатации.
- *Возможность защиты конфиденциальной информации:* в том случае, когда требования содержат конфиденциальную или секретную информацию, шаблоны могут быть использованы для защиты именно той части текста требования, доступ к которой должен быть защищен.

Последний пункт, несомненно, нуждается в некотором уточнении.
В оборонных и некоторых коммерческих проектах необходимо ограничивать доступ, но не ко всей информации, а только к некоторой ее части. Очень часто текст одного требования содержит информацию, относящуюся к различным уровням секретности.
Например, совершенно очевидно (не секретно), что современный военный корабль будет оснащен ракетами, которые он будет запускать, однако следующая информация о производительности может носить закрытый характер: возможное количество запусков в единицу времени, радиус поражения и т.д.
Вместо того, чтобы ограничивать доступ к целому требованию только из-за того, что некоторая его часть является конфиденциальной, данный подход дает возможность разрешить просматривать всё требование, но без доступа к конфиденциальной информации, содержащейся в некоторых его атрибутах. В действительности, при использовании такого подхода различные участники проекта (с разным уровнем доступа) могут видеть разные наборы атрибутов.

Одним из самых сложных с точки зрения формулирования и, к сожалению, одним из наиболее распространенных типов требований являются ограничения. Так вот как раз для ограничений метод шаблонов существенным образом облегчает задачу формулировки требований.
Авторы предлагают следующий подход для формулирования ограничений:

1. В первую очередь, соберите все возможные требования.
2. Подготовьте список ограничений различных типов, которые могут встретиться при работе над вашим проектом.
   Если этот список основан на предыдущем опыте выполнения аналогичного проекта, то, в этом случае, у вас уже есть полный набор шаблонов с ограничениями, который вы можете использовать для описания ограничений текущего проекта. В противном случае, вам придется разрабатывать новые шаблоны.

[7] *прим. переводчика:* Необходимо также отметить один предполагаемый недостаток метода – для русского языка применение данного метода может осложняться необходимостью согласования падежей.

3. Применительно к каждому требованию рассмотрите полный перечень возможных ограничений из вашего списка и определите, какие именно из них должны быть применимы (зафиксированы) для этого требования.
   Для выполнения этой процедуры очень удобно использовать таблицу. Столбцы будут соответствовать различным типам ограничений, а строки – ограничениям.
   Если для требования необходимо добавить ограничение определенного типа, то в соответствующей ячейке необходимо сформулировать это ограничение; если же необходимости в ограничении вообще нет, то в соответствующей ячейке нужно поставить «нет» (или N/A = not available).
4. Для каждого ограничения необходимо выбрать наиболее подходящий для него шаблон и сформулировать требование с его помощью.
5. Процесс заканчивается в тот момент, когда все ячейки таблицы заполнены.

Использование такого подхода позволяет ответить на два часто задаваемых вопроса:

- Как формулировать требование, в котором должно быть ограничение?
  (*Ответ:* использовать шаблоны)
- Как убедиться в том, что все ограничения учтены?
  (*Ответ:* использовать таблицу для анализа покрытия требований с ограничениями)

В таблице 4.4 показан пример набора шаблонов для требований с ограничениями.
Обратите внимание на тот факт, что для одного типа ограничений могут использоваться различные шаблоны, а так же на то, что ограничения могут иметь сложную классификацию. В приведенных примерах только текст, выделенный жирным шрифтом, относится непосредственно к самому ограничению.

Таблица 4.4 Примеры шаблонов требований с ограничением

Тип ограничения	Шаблон
Производительность/возможность	<Система> должна <выполняемая функция> <объект> **не менее чем <производительность> раз в <единица измерения>**
Производительность/возможность	<Система> должна <выполняемая функция> <объект> **типа <характеристика> в течение <производительность> <единица измерения>**
Производительность/мощность	<Система> должна <выполняемая функция> **не менее чем <количество>** <объект>
Производительность/своевременность	<Система> должна <выполняемая функция> <объект> **в течение <производительность> <единица измерения> с момента <событие>**
Производительность/периодичность	<Система> должна <выполняемая функция> **не менее чем <количество>** <объект> **в течение <производительность> <единица измерения>**
Способность к взаимодействию/мощность	<Система> должна <выполняемая функция> <объект> **состоящий из не менее чем <производительность> <единица измерения>с <внешняя сущность>**
Устойчивость/периодичность	<Система> должна <выполняемая функция> <объект> **с <производительность> <единица измерения> каждые <производительность> <единица измерения>**
Окружение/работоспособность	<Система> должна <выполняемая функция> <объект> **функционируя в <условия эксплуатации>**

4.10 Детализация требований

Использование шаблонов для «конструирования» требований позволяет практиковать следующее - некоторые ограничения или показатели производительности могут формулироваться и существовать как отдельные подпункты (подклассы) соответствующих функциональных требований.
Такой подход позволяет выделять эти подклассы в виде отдельных требований и устанавливать для них связи с соответствующими функциональными требованиям.

Вот тут то, как раз и возникает вопрос о степени (глубине) детализации информации. До какой же степени мы собираемся «расщеплять атом» при разработке требований?

На этот вопрос можно ответить следующим образом: требование может дробиться на подпункты до тех пор, пока средство поддержки работы с требованиями (Requirements Management tool) дает вам возможность видеть каждое требование в нужном контексте.
На рис. 4.4 представлен способ разделения требования, при котором подпункты становятся дочерними требованиями основного требования, образуя, таким образом, определенную иерархию. При этом главное требование остается пригодным для чтения и понимания само по себе, в то время как дочерние требования могут рассматриваться только в контексте «родительского» требования, хотя в отношении трассировки и связей они могут существовать вполне автономно.

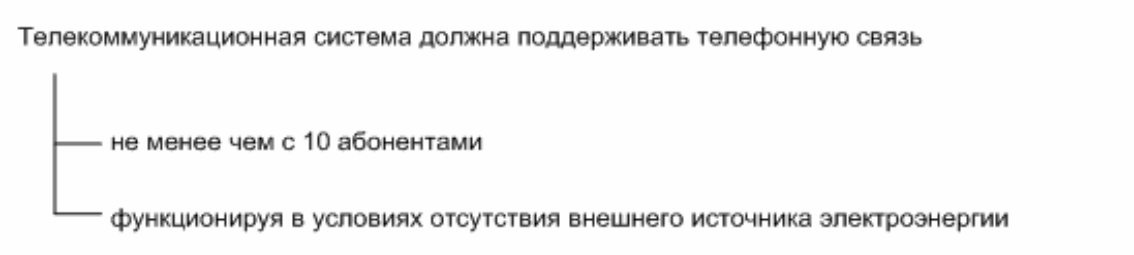


Рис. 4.4 Показатели производительности и ограничения, как подпункты.

Другими словами, вы можете работать с дочерним требованием как с вполне законченным элементом, пригодным для трассировки, но цитировать (увязывать) дочернее требование лучше все-таки вместе с текстом родительского требования.
Если вновь обратиться к примеру на рис. 4.4, выделив курсивом текст родительского требования в тексте дочерних требований, то имеется в виду следующее:

- Телекоммуникационная система должна обеспечивать телефонную связь
- *Телекоммуникационная система должна обеспечивать телефонную связь* не менее чем с 10 абонентами.
- *Телекоммуникационная система должна обеспечивать телефонную связь*, функционируя в условиях отсутствия внешнего источника электропитания.

Существуют различные способы организации иерархии таких подклассов. Предположим, что необходимо описать несколько разных возможностей, которые должны быть доступны «в условиях отсутствия внешнего источника электроэнергии».

Вариант организации требований представлен на рис. 4.5.

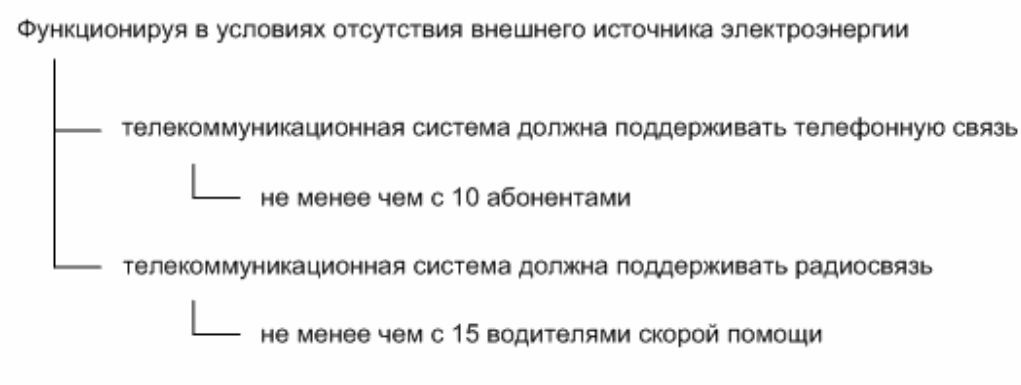


Рис. 4.5 Альтернативное представление подпунктов.

Для этого примера взаимосвязь между требованиями будет выглядеть иначе:

- *В условиях отсутствия внешнего источника электроэнергии* телекоммуникационная система должна поддерживать телефонную связь.
- *В условиях отсутствия внешнего источника электроэнергии телекоммуникационная система должна поддерживать телефонную связь*, не менее чем с 10 абонентами.
- *В условиях отсутствия внешнего источника электроэнергии* телекоммуникационная система должна поддерживать радиосвязь.
- *В условиях отсутствия внешнего источника электроэнергии телекоммуникационная система должна поддерживать радиосвязь*, не менее чем с 15 водителями скорой помощи.

Другими словами суть такого подхода состоит в том, что при построении иерархии требований (их детализации) родительское требование должно формулироваться таким образом, чтобы обеспечивать полный смысловой контекст для каждого дочернего требования, включая и возможную последующую детализацию дочерних требований на более мелкие блоки информации.

4.11 Критерии для написания текста требований

Независимо от языковых аспектов написания требований, существуют четкие критерии, которым должна удовлетворять формулировка ***каждого*** требования. Вкратце эти критерии можно сформулировать следующим образом:

- *атомарность*: каждое утверждение (формулировка требования) должно представлять собой один элемент иерархии, пригодный для установки связей с ним;
- *уникальность*: каждое требование должно иметь собственный уникальный идентификатор;

- *выполнимость*: требование должны быть технически реализуемо в установленные сроки, в рамках выделенного бюджета;
- *законность*: требование не должно противоречить применимому законодательству;
- *ясность*: требование должно быть понятно сформулировано (исключать неоднозначное толкование);
- *точность*: требование должно быть точным и лаконичным;
- *проверяемость*: должна существовать возможность проверки реализации каждого конкретного требования;
- *абстрактность*: формулировка не должна навязывать определенные технические решения, характерные для более низких уровней требований (спецификаций).

Дополнительно можно привести критерии, применимые к *набору* требований:

- *полнота*: все необходимые требования зафиксированы;
- *непротиворечивость*: не существует требований, противоречащих друг другу;
- *отсутствие избыточности*: каждое требование сформулировано только один раз (нет повторов);
- *модульность*: требования, близкие друг другу по смыслу, содержаться в одном разделе;
- *структурированность*: наличие ясной и четкой структуры документа с требованиями;
- *удовлетворенность*: достигнут требуемый уровень покрытия требований связями типа «удовлетворяется (посредством)»
- *тестируемость*: достигнут требуемый уровень покрытия требований тестами.

Для иллюстрации того, как ***не надо*** делать, ниже приводятся два «жутких» примера формулирования требований:

1. Система должна обеспечивать максимальный уровень производительности в течение всего времени работы, за исключением аварийных ситуаций, при которых она должна обеспечивать уровень производительности до 125%, но только если аварийная ситуации не длится более чем 15 минут, - в противном случае система должна уменьшить уровень производительности до 105%; но в случае, если удается достигнуть уровня производительности только 95%, система должна активировать режим «исключительно малого уровня» и поддерживать этот уровень в пределах 10% от начального значения в течение, как минимум, 30 минут.
2. Система должна обеспечивать основные функции текстового редактора, удобные для использования необученным персоналом, и должна работать в условиях «тонкого» Ethernet’а, проложенного по воздушной системе кабельных каналов с интегрированными сетевыми адаптерами, поставляемыми с дополнительными модулями памяти, при необходимости.

Эти примеры иллюстрируют классические **негативные** ситуации, характерные для разработки требований. Для того чтобы избежать этих ошибок, мы рекомендуем следовать простым правилам:

- *избегать хаоса*: формулируя требование, необходимо сконцентрироваться на самом важном; требование не должно быть похоже на роман;
- *избегать «лазеек»*: например, таких выражений, как «если это необходимо», поскольку такие «лазейки» делают требование неоднозначным и зачастую бесполезным;
- *избегать размещения более одного требования в один параграф*: зачастую, наличие в одном параграфе более одного требования легко определить по наличию союза «и»;
- *избегать рассуждений*;
- *избегать «размытых» понятий и слов*: обычно, в основном, часто, нормально, типично;
- *избегать использования неопределенных терминов*: например, удобный в использовании, универсальный, гибкий;
- *избегать принятия желаемого за требуемое*: напр., 100% надежный, приятный для всех пользователей, безопасный, подходящий для всех платформ, не должен никогда ломаться, обрабатывать все неожиданные сбои, быть готов к модернизациям для любых ситуаций, которые могут возникнуть в будущем и т.д.

Анализ первого примера показывает, что вместо одного требования, нужно писать 12. Развивая эту мысль, лучше всего выделить 4 отдельных условия эксплуатации - нормальное, аварийное, аварийное более 15 минут, режим «исключительно малого уровня», - и описать требования для каждого из этих условий.
Обратите внимание на имеющуюся лазейку во втором примере. Совершенно непонятен объем требования. Например, это требование можно интерпретировать и так: «Система должна обеспечивать основные функции текстового редактора … при необходимости».
Так требуется ли, в конце концов, текстовый редактор или нет? – старайтесь избегать таких ситуаций.

4.12 Заключение

Один из самых трудных моментов в разработке требований это начало – что называется, *«лиха беда-начало»*.
Разумеется, наличие разного рода методик по конструированию требований и умение ими пользоваться играют важную роль, но также весьма важно и фиксировать каждое требование с самого первого дня работы над проектом, и давать их для анализа и рецензирования другим участникам проекта.
Авторы предлагают использовать следующую «безопасную» последовательность шагов в процессе работы с требованиями:

- Определите структуру требований и пытайтесь совершенствовать ее в процессе работы с требованиями.
- Фиксируйте (записывайте) требования как можно раньше, даже если они весьма далеки от совершенства.
- Заранее определите атрибуты требований, которые в последующем будут использоваться для классификации и детализации требований.

- Как можно быстрее подготовьте первый вариант требований, для того чтобы стимулировать получение отзывов.
- Постоянно совершенствуйте (улучшайте) требования в процессе работы, удаляя повторения, преждевременные и недозволенные технические решения, противоречивость.
- Постоянно обсуждайте требования, собирайте замечания и, не откладывая в долгий ящик, корректируйте требования, создавая новые их версии (принцип «мозгового штурма»).
- Демонстрация пользователям ваших наработок гораздо лучше, чем «экспертный анализ специалистов».

А для написания (формулирования) требований можно следовать нижеприведенным правилам:

- Используйте простой и прямой (однозначный) язык;
- Пишите такие требования, которые могут быть проверены;
- Используйте определенную и согласованную терминологию;
- В *одном* требовании формулируйте только *одно* утверждение.
- Начав работать над одним требованием, постарайтесь закончить с ним прежде, чем переходить к другому.

5 Разработка требований в области проблем

Дело совсем не в том, что они не видят решения проблемы.
Дело в том, что они не видят саму проблему.
Гилберт Кит Честертон (Gilbert Keith Chesterton),
писатель, 1874–1936

5.1 Что такое область проблем?

Под областью проблем подразумевается то окружение, область или среда, в которой система будет использоваться. Вот почему так важно рассматривать требования с эксплуатационной точки зрения - ведь предназначение любой системы или любого продукта это позволить человеку или другому устройству выполнять требуемую функцию. Вот эта самая необходимость «обеспечить функционирование системы» и есть ключевая идея разработки требований в области проблем.
Столкнувшись с необходимостью сбора требований, возможно, вам захочется спросить потенциального пользователя:

Что вы хотите, чтобы эта система делала ?

Часть потенциальных пользователей этот вопрос поставит в тупик, а у другой части пользователей найдется мало что ответить.
Если ваши потенциальные пользователи уже используют какие-либо системы, то у них обычно существуют идеи на тему *как их улучшить*. В случае если никаких систем у потенциальных пользователей сейчас нет, то и почвы для подобных идей у них тоже нет.
В любом случае, в качестве ответа на свой вопрос вы получите описание того или иного решения, потому что ваш вопрос уже предусматривал в качестве ответа описание функциональности рассматриваемой системы.

Для того чтобы в самом начале избежать подобных прыжков в область решений, необходимо задать другой вопрос:

Зачем вам нужна эта система?

или

Какие задачи должна решать необходимая вам система?

Как только речь заходит о задачах, которые должна будет решать рассматриваемая система, люди сразу же начинают думать о том, ***что*** они будут делать с помощью системы, вместо того чтобы думать о том, ***как*** они это будут делать.
Однако все то, что люди хотят получить с помощью системы, может быть сформулировано даже без описания возможных решений. И это весьма важный момент, потому как, избегая уклона в сторону решения, мы оставляем пространство для работы архитекторов и системных инженеров.

Вот почему некоторые специалисты даже настаивают на том, что слово «система» должно быть полностью исключено из предыдущего вопроса, а сам вопрос должен звучать следующим образом:

*Что вы хотите, чтобы **вы** могли делать?*

Ответ на этот вопрос должен начинаться фразой:

Я хотел бы иметь возможность …

Именно такой ответ является требованием типа «возможность» (описывающим возможность). И именно этот тип требований является ключевым в области проблем.

Таким образом, установив, что при разработке требований в области проблем первичной задачей является выявление возможностей, мы формулируем для самих себя следующий вопрос:

А кого, собственно, нужно опрашивать?

Поиск ответа на этот вопрос приводит нас к необходимости идентифицировать некую группу людей, которую мы будем называть «заинтересованные стороны» (stakeholders). Под термином «заинтересованные стороны» могут пониматься либо непосредственно люди, либо организации, прямо или косвенно заинтересованные в существовании системы.

Наконец, мы должны задуматься над тем, какие типы моделей могут быть пригодны для моделирования проблемной области. Необходимо отметить, что любые модели, которые используются для описания проблемной области, должны быть, в первую очередь, понятны именно заинтересованным сторонам, поскольку они будут отвечать за их анализ, оценку и утверждение (принятие). А поскольку специалисты заинтересованных сторон отбираются по принципу их набольшей компетентности и опыта в проблемной области, то, зачастую, они не желают или не могут работать с моделями, которые кажутся им хоть немного «техническими».

К примеру, если вы собираетесь посетить автосалон для того, чтобы ознакомиться с новыми автомобилями, их внешним видом, убранством салона и т.д., то вас вряд ли заинтересуют графики, поясняющие очередность впрыска топлива в цилиндры двигателя, или рассуждения на тему принципов построения системы управления двигателем. Вероятнее всего вас будут интересовать мощность двигателя, время разгона до скорости 100 км/ч, расход горючего на 100 км, комфортабельность и комплектация автомобиля. Другими словами, вы захотите представить себе - какой машина будет в управлении, как бы хорошо вы смотрелись в ней, отправляясь в поездку с подругой или другом. Представляя себе эту поездку, вы начнете думать обо всех тех особенностях и преимуществах автомобиля, которые были бы полезны вам в поездке. Вот это и есть простой пример сценария использования.

Давно было подмечено, что сценарии использования – это очень хороший способ моделирования того, что люди делают, или того, что они хотели бы иметь возможность делать. Сценарии во многом соответствует тому, как люди думают о своей работе и о своих проблемах. Сценарий может быть разработан совместно с заинтересованной стороной и в дальнейшем использоваться как основа для обсуждения требуемых возможностей.

Заключительным аспектом разработки требований в проблемной области является выявление имеющихся ограничений. Например, при покупке автомобиля, вы будете иметь лишь определенную сумму денег на покупку, или захотите, чтобы автомобиль был в вашем распоряжении уже к конкретной дате.

После того, как выше мы определили все ключевые моменты, у нас появляется возможность проиллюстрировать основной (общий) процесс разработки требований из области проблем.

5.2 Определение основного процесса

На рис. 5.1 представлен основной (общий) процесс разработки пользовательских требований (stakeholder requirements).

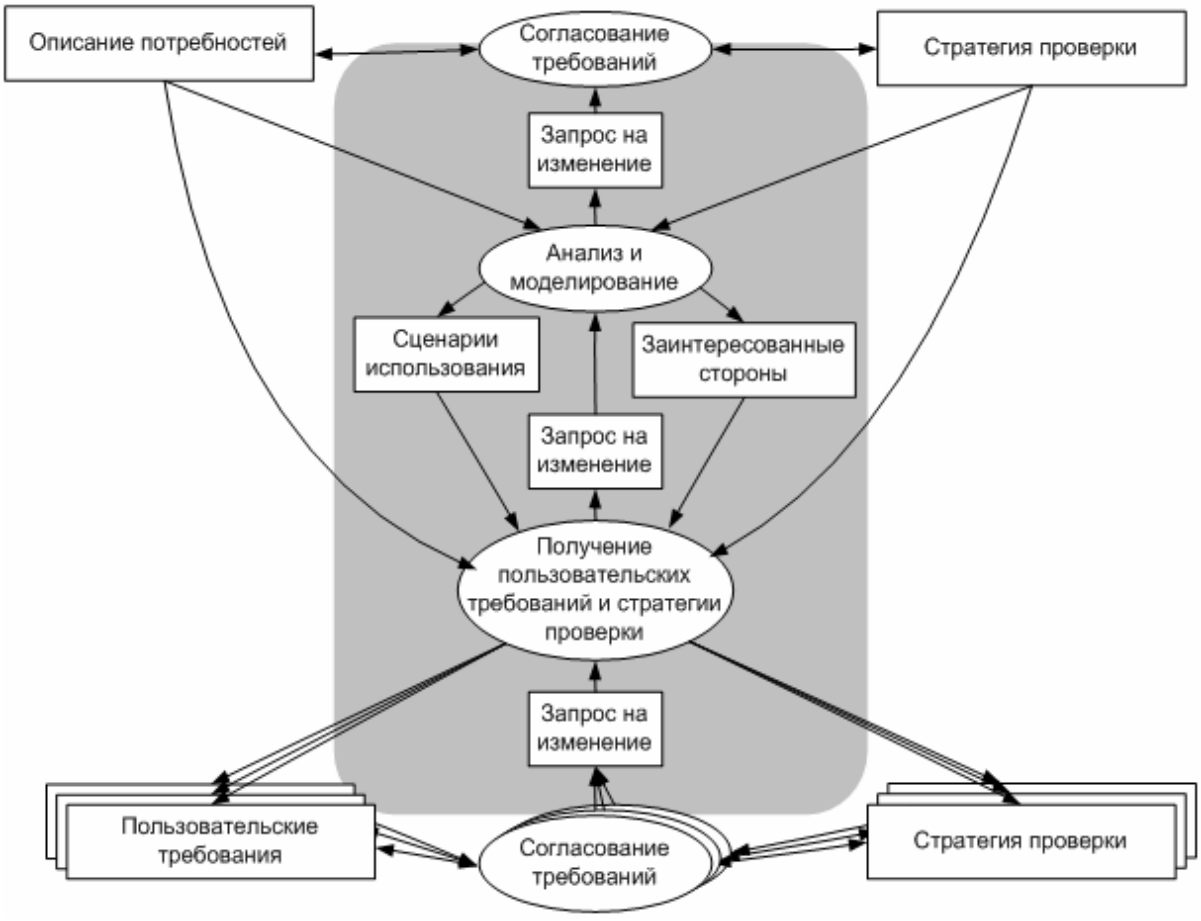


Рис. 5.1 Процесс разработки пользовательских требований.

Начальной точкой, инициирующей процесс, является описание потребностей.

Это может быть даже что-то совсем небольшое по содержанию, например, короткая служебная записка руководителя компании техническому директору, в которой говориться о том, что компании необходим новый продукт для того, чтобы быть на шаг впереди

конкурентов. Или же, наоборот, - представьте, что компания уже проводила исследования для определения возможных альтернативных решений существующих проблем и полученные результаты сформулированы в виде документа, из которого уже даже можно получить часть сценариев использования.

На рис. 5.1 показано, что процесс *анализа и моделирования* создает набор сценариев использования и формирует список *заинтересованных сторон*. *Пользовательские требования* при этом будут производными от требований *заинтересованных сторон*.

Анализ и моделирование, получение пользовательских требований, стратегии проверки для них будут рассмотрены более подробно в последующих разделах.

5.3 Согласование требований с заказчиком

Зачастую процесс согласования пользовательских требований в начале проекта носит неформальный характер. Обычно документ, содержащий потребности, пишется в простой свободной форме и совершенно не похож на требования.

Другими словами, этот документ обычно содержит достаточно размытое описание потребностей вперемешку с некоторым пояснительным текстом. Он не содержит четко выписанных детальных требований, к которым могли бы впоследствии приходить входящие связи типа «удовлетворяет».

Поэтому процесс получения пользовательских требований отличается от процесса конструирования требований на следующих уровнях, так как первый начинается с весьма нечетких и бесформенных формулировок (вплоть до «сделайте мне красиво»).

Одним из ключевых аспектов формирования пользовательских требований является определение границ будущей системы (границ ответственности). Обычно эти границы фиксируются с самого начала при определении наборов сценариев использования.

5.4 Анализ и моделирование

На рис. 5.2 показано, что процесс анализа и моделирования для разработки требований является «инициативным» для области проблем.

В начале определяются все заинтересованные лица, а затем вместе с ними создаются сценарии использования.

5.4.1 Определение заинтересованных сторон

Ранее мы становили, что заинтересованными сторонами могут быть люди или организации, которые имеют определенное отношение к разрабатываемой системе, например, либо влияют на нее и взаимодействуют с ней, либо система влияет и воздействует на них.

При этом типы заинтересованных сторон могут различаться в зависимости от характера и предназначения системы, сравните, например, две системы – одну, используемую как товар широкого потребления, и другую - для управления воздушным транспортом или железной дороги.

Люди, имеющие свое мнение относительно создаваемой системы, включают и тех, кто будет самым непосредственным образом использовать систему (работать с ней).

Заметим при этом, что этими людьми могут быть не только непосредственные пользователи системы, например, пассажиры самолета или поезда, но и люди, которые не являются пассажирами, но могут, например, пострадать в случае аварии самолета или поезда. А группа лиц, которая несет ответственность за систему, может включать и руководящий (управляющий) персонал, и персонал, отвечающий за безопасность.

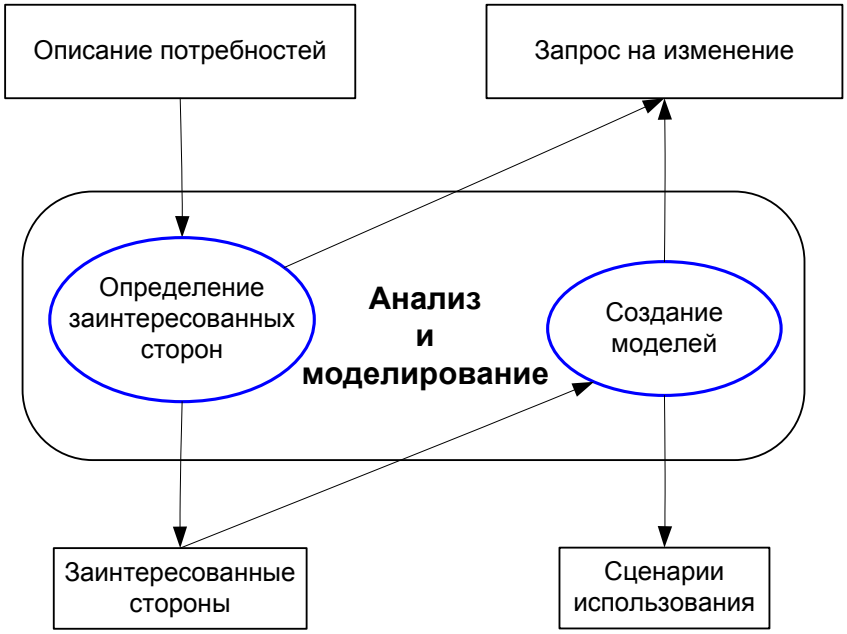


Рис. 5.2 Процесс анализа и моделирования для разработки пользовательских требований.

В качестве примера, ниже приводится список возможных категорий заинтересованных сторон, который может использоваться как основа при разработке вашего собственного списка с целью проверки его полноты. Этот список категорий не претендует на абсолютную полноту, однако может помочь при мозговом штурме для определения всех заинтересованных сторон.

- **Руководство**: люди, отвечающие либо за бюджет разработки, либо за бюджет организации (департамента), в которой система будет функционировать. Хорошей идеей является также привлечение людей, ответственных за стратегию развития компании, для того чтобы получить и их мнение о том, насколько разрабатываемая система соответствует задачам, стоящим пред компанией.
- **Инвесторы**: люди, приглашенные к финансированию или уже вложившие деньги в разработку системы или организацию, разрабатывающую систему, или же в организацию, в которой система должна внедряться.
- **Пользователи системы**: естественно, что эта группа людей является очень важной. Они непосредственно заинтересованы в тех возможностях или сервисах, которые должна предоставлять новая система. Необходимо также отметить, что при этом весьма возможно существование таких пользователей, которые не будут взаимодействовать с системой напрямую. Например, пользователями телескопа Хаббла являются астрономы. Они направляют запросы на фотоснимки определенных районов и получают их, как

только снимки готовы, но они сами непосредственно не управляют телескопом. Пользователи существующих систем являются очень полезным источником знаний о проблемах, присутствующих в этих системах. Они могут дать бесценные советы относительно того, каким образом может быть улучшена разрабатываемая система.

- **Обслуживающий персонал**: не смотря на то, что их прямой обязанностью является поддержка системы уже после ее запуска, обслуживающий персонал имеет важные требования к системе, которые необходимо учитывать для того, чтобы облегчить обсуживающему персоналу поддержку и обслуживание будущей системы.
- **Утилизаторы**: эта сторона играет важную роль в соблюдении требований законодательства по защите окружающей среды. Требования этой стороны могут существенным образом повлиять на проект системы, особенно на выбор предполагаемых к использованию материалов.
- **Обучающий персонал**: так же как и обслуживающий персонал, эти люди заинтересованы в том, чтобы система была простой в использовании, а, соответственно, и легкой для обучения. Эти люди могут требовать, чтобы система обеспечивала такой режим функционирования, при котором одновременная работа с реальными данными и данными, предназначенными для обучения, не нарушала бы работоспособность и безопасность всей системы.
- **Покупатели**: для больших систем, особенно широко используемых в промышленности и транспорте, весьма характерно, что люди, которые платят за систему (покупатели-заказчики), не участвуют в ее разработке и последующей эксплуатации. Однако они играют большую роль в определении объема требований к системе с точки зрения отношения затрат к получаемым преимуществам. Хотя при разработке оборудования, покупатель может также являться и непосредственными его пользователем, например, сотового телефона или автомобиля.
- **Маркетинг и продавцы**: эти люди играют ключевую роль в определении возможностей системы; особенно при разработке продуктов (оборудования) массового потребления, когда узнать мнение всех потенциальных пользователей просто невозможно.
- **Эксперты по эргономике и эффективности**: эти люди знают, как оптимизировать систему, для того чтобы сделать ее использование более эффективным. Ими учитываются факторы эргономики, легкости изучения и освоения, возможности надежного использования в экстренных ситуациях или условиях интенсивной эксплуатации (например, для систем управления воздушным транспортом).
- **Эксперты по области применения**: обычно новая система разрабатывается для работы не в «чистом поле» и уже существуют системы, с которыми она должна взаимодействовать. При этом возможно также существование косвенно влияющих аспектов внешнего окружения, таких как контроль за загрязнением окружающей среды (ваша система не должна загрязнять окружающую среду), или когда необходимо обеспечить устойчивость системы к действиям окружающей среды (например, сложные погодные условие, или погружение на глубину).
- **Правительство**: законы, правила, инструкции и многие другие официальные документы органов власти регулируют и определяют то, что система может делать, а что ей категорически запрещено.
- **Стандартизирующие органы**: на разрабатываемую систему могут влиять существующие или разрабатываемые стандарты, которые могут быть международными (например, стандарт сотовой связи GSM), национальными или внутри корпоративными.

- **Общественное мнение**: в разных странах мира существуют различные мнения по одному и тому же вопросу. Поэтому при разработке продуктов, предназначенных для поставки на экспорт, необходимо принимать во внимание географический и национальный факторы.
- **Регулирующие органы**: для того, чтобы получить необходимый сертификат соответствия, эти организации могут настаивать на том, чтобы создаваемая система (продукт) имела определенную прозрачность (открытость). В качестве примера можно привести Министерство железнодорожного транспорта Великобритании (Rail Regulator) и Управление по контролю за продуктами и лекарствами (FDA) в Соединенных Штатах.

Получив как можно более полный список категорий потенциальных заинтересованных сторон, следует принять решение о том, какие именно из этих категорий должны участвовать в постановке требований и как их вовлечь в этот процесс.
В некоторых случаях возможен прямой контакт с представителями одной из заинтересованных сторон (категорий), например, с потенциальными пользователями системы. Тогда стоит определить конкретных представителей, с которыми вы будете общаться.
В других случаях прямой контакт в принципе невозможен, например, если заинтересованная сторона – общество. Тогда представляется вполне возможным назначить на эту роль кого-то из вашей команды и прислушаться к его мнению.
В конце концов, на выходе этого процесса мы получаем список представителей заинтересованных сторон (см. рис. 5.2).

5.4.2 Разработка сценариев использования

В любой дискуссии большинство разговоров ведется вокруг определенных предположений и допущений, с которыми собеседники согласны. Совокупность этих предположений и допущений может рассматриваться как модель их взаимопонимания. Любая попытка начать обсуждение требований в отсутствие хотя бы минимального понимания будет весьма непродуктивной.

Одним из основных организующих механизмов, помогающих обсуждению пользовательских требований (возможностей), является работа со сценариями функционирования или использования. Такой подход синтезирует структуру, которая упорядочена временем. При этом применение сценариев при разработке пользовательских требований дает возможность использовать их нотацию для установления общей системы взглядов, в рамках и контексте которой последующий диалог с пользователем становится более продуктивным.

Сценарии стимулируют человека (представителя заинтересованной стороны) задумываться не только о том, как он в данный момент выполняет свою работу, но и о том, как бы он хотел ее делать в будущем. Это ведет к тому, что люди многократно «проигрывают» (репетируют) в уме разные сценарии того, как они хотели бы это делать. После того, как сценарий в какой-то степени согласован (проговорен), можно приступать к формулировке индивидуальных первичных требований с тем, чтобы уже более точно определить, что же каждый из пользователей хотел бы иметь возможность делать в каждой отдельной точке сценария.

Сценарии являются великолепным механизмом для выявления пользовательских требований - они заставляют сосредоточиться именно на тех целях, которых хотят достичь заинтересованные стороны. Любой проработанный сценарий представляет собой совокупность *последовательных результатов* (или достигнутых состояний), полученных *в хронологическом порядке*.
Как показано на рис. 5.3, сценарий может быть представлен последовательностью целей и демонстрировать возможности, которые обеспечивает система заинтересованной стороне, но без расшифровки того, каким образом эти цели могут быть достигнуты. Другим словами, сценарий представляет собой иерархию возможностей.

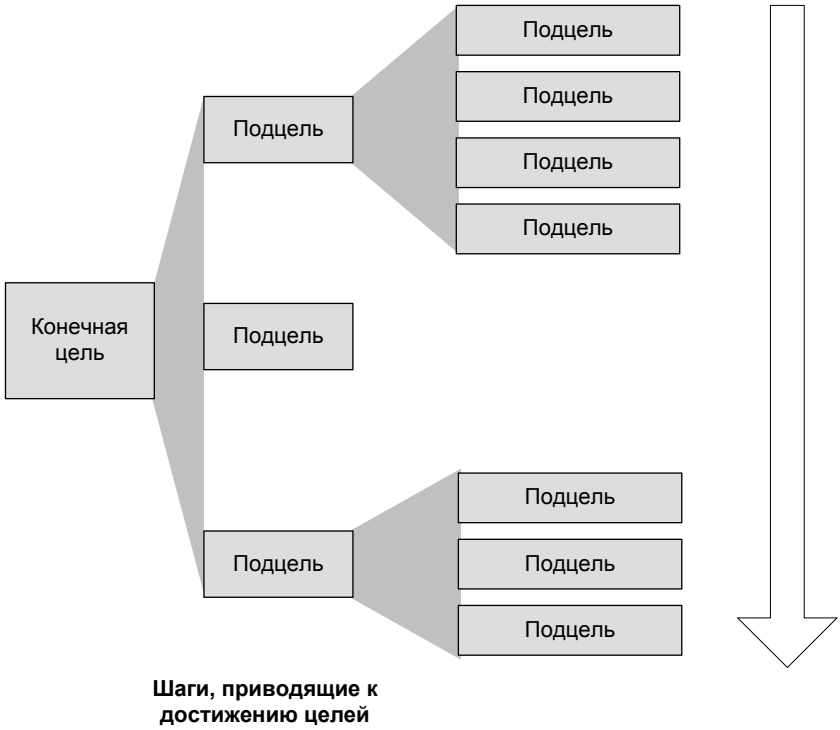


Рис. 5.3 Сценарий использования как иерархия целей.

Хронологическая последовательность получения сценария позволяет в дальнейшем неоднократно проигрывать возможные варианты функционирования системы. Заинтересованные стороны могут шаг за шагом проверять этот функционал и искать недостающие или дублирующие элементы. Такой подход позволяет избежать принятия преждевременных соглашений о конкретных решениях для достижения поставленных целей, а сосредоточиться только на определении проблем.

Существует вполне определенный подход, которому можно следовать при разработке сценариев использования. Основной вопрос, который нужно задавать представителю заинтересованной стороны это: «Чего вы хотите достичь?» или «В каком состоянии вы хотите находиться?». Ответом на этот вопрос будет характеристика конечного состояния, которого необходимо достичь.

Далее, задавая уточняющие вопросы, необходимо получить описание промежуточных целей (состояний) или шагов, следование по которым и приводит к достижению конечной цели. Совокупность этих промежуточных состояний и рассматривается затем как иерархия (дерево) целей.
Таким образом, вырисовывается следующий порядок действий для получения сценария использования:

- Начать с получения характеристики конечной цели (состояния).
- Установить необходимые возможности (пользовательские требования) для достижения конечной цели.
- Разбить крупные шаги на более мелкие (промежуточные).
- Строго соблюдать иерархическую структуру.
- Перепроверять информацию на каждом этапе.
- Избегать описания конкретных решений.

Если представитель заинтересованной стороны затрудняется описать промежуточные шаги, то необходимо попросить его описать типичную ситуацию, чтобы узнать что будет делать человек в этой ситуации, и такими образом получить требуемый результат. Если же разрабатывается совершенно новая система, то ему придется напрячь все свое воображение. Таким образом, мы можем получить полное представление того, что по ожиданиям человека будет происходить или что человек ожидает получить на каждом из промежуточных шагов. На этом пути важно выявлять такие этапы, которые могут быть необязательными или повторяющимися. При этом следует постоянно искать ответ на вопрос – будут ли отличающиеся внешние условия влиять на последовательность шагов.

Необходимо определить последовательность шагов, а также постоянна (фиксирована) эта последовательность или она может изменяться; при этом – если она может изменяться - необходимо определить те факторы или условия, под действием которых это происходит. Например, если вы задумаете нарисовать картину, вам понадобятся бумага или холст, краски, кисти и т.д., но совершенно неважно, что именно появится у вас в первую очередь. Это дает возможность выполнять некоторые действия в произвольной последовательности или делать что-то параллельно.

Как на любом этапе получения требований очень важно воспринимать (и фиксировать) все, что говорит представитель заинтересованной стороны. В дальнейшем все это может быть пересмотрено и уточнено. Рекомендуется почаще просить более подробно объяснить, что именно в данном конкретном случае имеется в виду.

Сценарии отражают возможности (в терминах из области проблем), которые должна обеспечивать система и которые выстроены иерархическим образом, однако они не объясняют каким образом система должна обеспечивать эти возможности.
Можно отметить следующие преимущества использования сценариев:

- позволяют представителям заинтересованных сторон пошагово проигрывать сценарии функционирования;
- позволяют находить пропущенные шаги;
- различные заинтересованные стороны могут предлагать различные сценарии;
- позволяют строить хронологические последовательности.

Характеристики сценариев использования

На рис. 5.4 приведен пример сценария, описывающего проведение выходного дня на парусной лодке, которую можно транспортировать на легковом автомобиле.

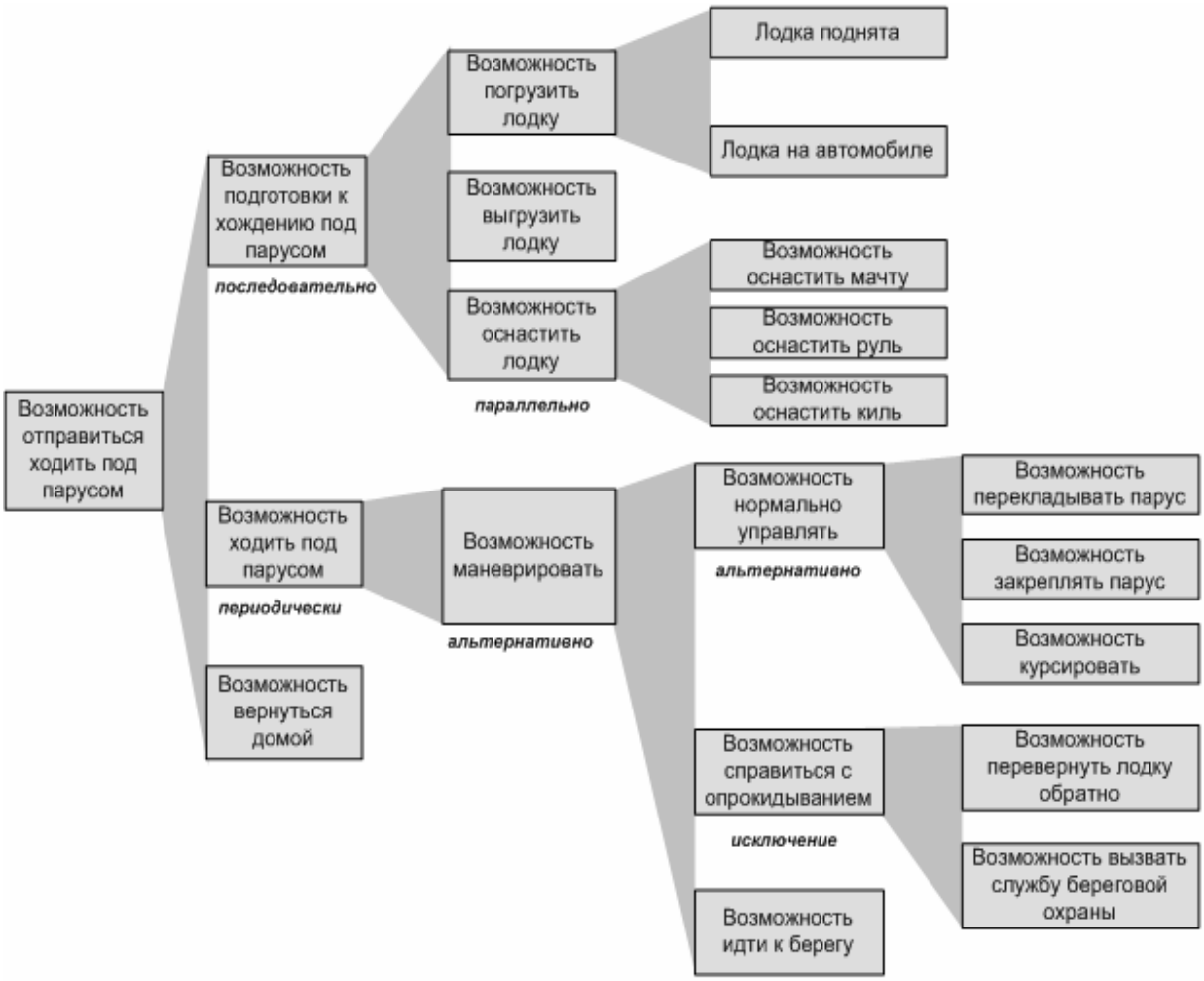


Рис. 5.4 Пример сценария использования.

Этот сценарий покрывает все моменты путешествия, начиная от погрузки лодки на автомобиль, подготовку к отплытию, хождение на лодке под парусом и возвращение домой. Сценарий также иллюстрирует и другие аспекты:

- временную последовательность действий в общих чертах;
- узловые элементы сценария являются возможностями верхнего уровня;
- альтернативы;
- периодически повторяющееся поведение;
- моменты, когда последовательность действий не имеет значения (параллельные ветви);
- исключения.

Использование временных последовательностей играет важную роль. Это помогает заинтересованным сторонам не только понять сценарий использования вообще, но и помогает формулировать пользовательские требования в правильном контексте.

Очень важно, что описания всех узловых элементов сформулированы как возможности соответствующего уровня. Использование слова «*возможность …*» в имени узлового элемента помогает избежать тенденции описывать возможность как функцию (а, следовательно, избежать описывать далее ее детальную реализацию).

Сценарии обеспечивают весьма мощную поддержку при выявлении исключений. Как известно, во многих системах функциональность по обработке исключений бывает гораздо более сложной, чем та функциональность, которая обеспечивает их обычные возможности, необходимые заинтересованным сторонам. Можно попытаться попросить заинтересованные стороны самим сформулировать такие исключения, задавая им вопросы, типа: «*какое развитие событий может считаться ошибочным при выходе из данного состоянии?*» или «*какое развитие событий может считаться ошибочным при входе в данное состояние?*». Естественно, что описание действий по исправлению ошибок (возвращению в нормальное состояние) может быть получено с помощью вопроса: «*что должно произойти (быть сделано), если события начнут развиваться по «неправильному» сценарию?*».

Как видно из рис. 5.4, сценарий отражает возможность связи со службой береговой охраны в том случае, если лодка опрокинется. В отсутствии сценария это требование могло бы быть запросто пропущено.

Данный пример прекрасно иллюстрирует также и то, как сценарий может облегчить задачу выявления пропущенных областей требований. В данном случае из сценария следует, что были пропущены *возможность транспортировать погруженную лодку* (до места спуска на воду) и *возможность спустить лодку на воду*.

Целью создания сценариев является облегчение общения и взаимопонимания между людьми. Сценарий сам по себе не является требованием; в большей степени это просто упорядоченная структура для получения требований. Сценарии лишь помогают получить полный набор требований путем подробного рассмотрения всех функциональных аспектов использования будущей системы.

При этом следует заметить, что никакая из методик моделирования - и данная не исключение - не может отражать все возможно существующие концепции. Не существует единственно правильного пути, и разные специалисты предпочитают использовать различные методы и модели.

5.4.3 Определение границ системы

На начальном этапе разработки сценариев использования очень полезно слегка раздвинуть границы предполагаемой системы по сравнению с теми пределами, в рамках которых она будет функционировать. Это поможет убедиться, что система будет правильно вписана в окружающий контекст и что ничего не будет пропущено. Но на определенном этапе очень важно все-таки четко определить границы системы, а, следовательно, и объемы работ по проекту.

Такой точкой определения полного объема проекта и рамок системы может являться подборка и компоновка полного набора сценариев использования. Разумеется, в дальнейшем это решение может быть пересмотрено, например, после предварительной

оценки общей стоимости проекта (бюджета). Такая оценка обычно дается специалистами, имеющими опыт разработки систем в той же предметной области, что и разрабатываемая система. Конечно же, эти оценки, данные только на основе разработанных сценариев, являются весьма приблизительными, однако они дают достаточно хорошую возможность представить является ли адекватным бюджет предполагаемого проекта.

5.5 Получение требований

Для наилучшего восприятия изложения мы разделили *процесс получения требований* и *процесс формирования стратегии проверки*. Процесс получения требований мы опишем в этом разделе, а процесс формирования стратегии проверки в следующем.

Рис. 5.5 отражает процесс получения требований для области проблем.

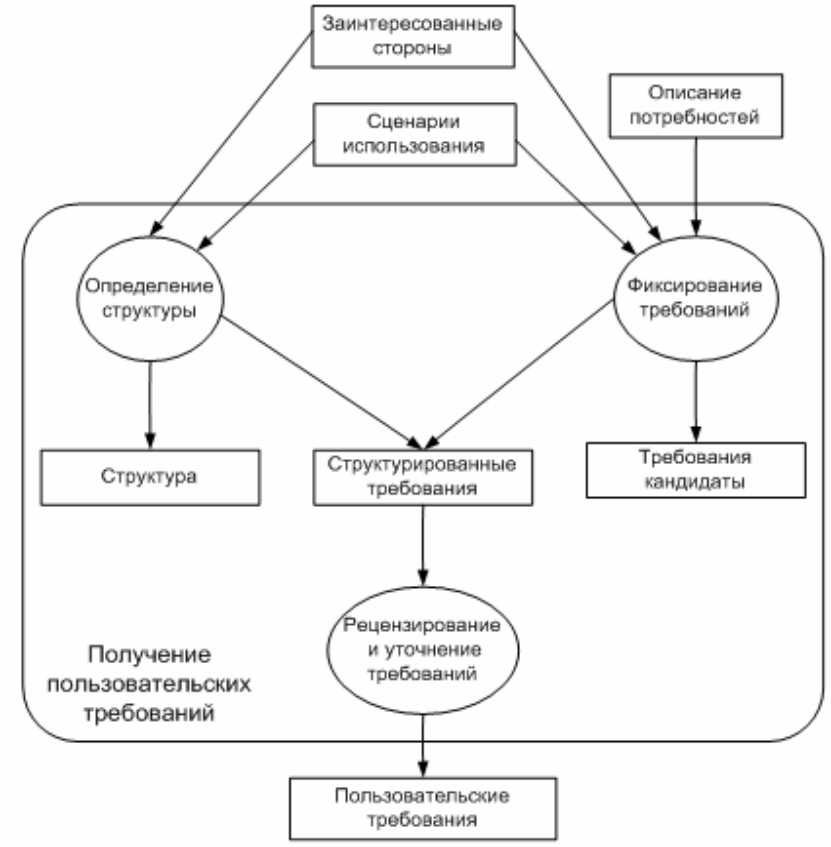


Рис. 5.5 Получение требований для области проблем.

Ключевыми аспектами этого процесса являются фиксирование требований и определение (формирование) структуры, в которой их следует размещать. После того, как определена структура для требований и сформулированы требования-кандидаты, становится возможным размещать требования в рамках структуры. На самом деле, конечно же, эти два процесса протекают параллельно и более того, - процесс получения требований-кандидатов вносит свою лепту в формирование структуры.

Следовательно, вместо того, чтобы иметь разнесенные процессы формирования структуры и получения требований-кандидатов, рис. 5.5 иллюстрирует, как оба эти процесса вносят каждый свой вклад в создание структурированных требований.

После получения структурированных требований желательно провести их последующее рецензирование, анализ и утонение.

5.5.1 Определение структуры

Наличие хорошо организованной структуры играет исключительно важную роль в обеспечении управления и координации работ на всем протяжении жизненного цикла проекта. Сбор пользовательских требований обычно является итеративным процессом - требования одно за другим фиксируются, уточняются, «очищаются» и затем помещаются в общую структуру.

При этом некоторые авторы думают по-другому и утверждают, что:

- пользовательские требования не поддаются структурированию по своей природе;
- вполне достаточно лишь установить связи между пользовательскими требованиями и спецификациями;
- мы никогда не видели законченной структуры, поэтому достаточно просто поочередно работать с каждым отдельным требованием (без какой-либо привязки к структуре).

Такие подходы не имеют ничего общего с качеством разработок, а являются всего лишь отражением временных интересов разработчиков.

Мы утверждаем, что требования обязательно должны быть организованы и должна обязательно существовать хорошая структура для управления каждым индивидуальным требованием по мере его появления и внесения в эту структуру. Что характерно, аргументы в пользу необходимости структурирования уже описаны в Главе 4, поскольку являются теми же, на которые уже обращалось внимание при обсуждении проблем работы с требованиями, как в области проблем, так и в области решений. Лишний раз подчеркивая особую важность построения хорошей, понятной структуры, эта глава большее внимание уделяет именно тому, как получить такую структуру для пользовательских требований.

Основной концепцией построения структуры требований является сценарий использования. Однако для сложных систем может существовать достаточно большое количество сценариев использования. В таком случае рекомендуется все-таки потратить время и усилия на то, чтобы попытаться соединить все имеющиеся сценарии в один всеобъемлющий, если такое возможно. Конечно же, это не всегда может удаться, но попробовать в любом случае стоит. Помимо достижения своей прямой цели этот процесс даст многим возможность задуматься о системе, как о некоем целом, представить ее общий функционал, и, возможно, выявить очень много проблем.

Чтобы проиллюстрировать каким образом сценарии могут иногда объединяться, приведем пример из области ресторанного бизнеса. Для описания ресторана могут быть использованы три следующих сценария, которые отображены на рис. 5.6 – 5.8.

- жизнь ресторана от начала до конца – сценарий владельца;
- рабочий день в ресторане – сценарий управляющего;
- посещение ресторана клиентом – сценарий клиента.

Первый сценарий описывает жизнь ресторана – вначале ресторан приобретается владельцем, затем следует период владения рестораном и, в заключении, наступает момент его продажи.

Рис. 5.6 Сценарий жизни ресторана.

Второй сценарий описывает все состояния, в которых ресторан находиться в течение рабочего дня.

Рис. 5.7 Сценарий рабочего дня в ресторане.

Первой целью (состоянием) является пополнение запасов продуктов и напитков. В данном случае подразумевается, что у ресторана несколько поставщиков, но порядок доставки продуктов и напитков не имеет значения. Конечно, можно поспорить на тему, что пополнение запасов обязательно должно заканчиваться до момента открытия ресторана, но в нашем примере мы упростили себе жизнь, чтобы сделать пример более наглядным, и будем считать, что вся доставка осуществляется до момента открытия ресторана.

В сценарии рабочего дня ресторана затем идут периоды, когда ресторан бывает открыт для посетителей и когда ресторан закрыт - завершен рабочий день, производится уборка и составляется список для пополнения запасов на следующий день.

Сценарий посещения ресторана клиентом представляет собой просто последовательность состояний.

Рис. 5.8 Сценарий посещения ресторана клиентом.

Если теперь мы попытаемся представить себе, каким образом можно объединить все эти отдельные сценарии в один общий, то можем заметить, что:

- сценарий «*рабочий день в ресторане*» является повторяющимся сценарием для состояния «*владение*» в сценарии «*жизнь ресторана*», и
- сценарий «*посещение ресторана клиентом*» может быть параллельно повторяющимся сценарием для состояния «*открыт*» в сценарии «*рабочий день в ресторане*».

На рис. 5.9 показана общая структура, объединяющая все три сценария.

Рис. 5.9 Объединенный сценарий и структура для получения требований для ресторана.

Этот объединенный сценарий может теперь рассматриваться как структура заголовков (структура разделов) для документа, содержащего требования.

Конечно же, не всегда есть возможность объединить все сценарии в один. Тут не существует универсального подхода, который мог бы всегда срабатывать. Если объединение не удается, то нужно просто рассматривать один сценарий за другим. И тогда структура документа с требованиями будет повторять последовательность сценариев, в каждый из которых «встроены» свои собственные требования.

По сути, структура в большой степени формируется в зависимости от списка заинтересованных сторон, участвующих в проекте. И хотя многие сценарии использования могут иметь различные ключевые цели (состояния), все равно следует предпринимать попытки включения сценариев один в другой.

В тоже время необходимо внимательно следить за тем, чтобы в структуре (документе, сценарии) не было повторений. Если же повторяющиеся части все же встречаются, то необходимо сделать так, чтобы этот элемент присутствовал только в одном месте.

Для этого можно использовать два подхода. Первый подход заключается в том, чтобы вообще исключить повторяющиеся разделы изо всех тех мест, где они присутствуют, и оформить (вынести) это в виде независимой секции. А уж потом изо всех тех мест, где этот элемент встречался ранее, сделать ссылки на новую секцию. Другой подход, заключается в

том, чтобы оставить этот раздел в том месте, где он встречается впервые, а вместо всех последующих повторений использовать ссылки на это место в структуре.

5.5.2 Сбор требований

Источники пользовательских требований

Пользовательские требования могут быть сформированы из большого количества источников, например:

- интервью с представителями заинтересованных сторон;
- сценарии использования (обычно получаемые в результате интервью);
- описательная документация (напр., различные отчеты, анализ рынка);
- действующие системы, которые необходимо модернизировать;
- проблемы, обнаруженные в существующих системах, и идеи как их исправить;
- опыт работы с аналогичными системами;
- разного рода прототипы, макеты, эскизы, т.д. продукта или самих требований;
- возможности новых технологий (согласованные с заинтересованными сторонами);
- результаты исследований;
- опросы;
- наблюдение за работой персонала или изучение видеозаписей их работы.

Интервью

Интервью это очень непростая задача. Для того чтобы суметь «вытянуть» из представителей заинтересованных сторон реальные требования, специалист, проводящий интервью, должен, в первую очередь, быть коммуникабельным и уметь правильно общаться. Интервью это исключительно психологическая задача, не имеющая ничего общего с организационными или техническими аспектами разработки систем. Необходимо помнить, что получение пользовательских требований от заинтересованных сторон это *человеческая*, а не техническая проблема. Однако несмотря на это, специалист, проводящий интервью, должен хорошо разбираться и в проблемной (предметной) области, чтобы не испытывать трудности при общении, и понимать все, что ему будут говорить представители заинтересованных сторон.

Очень важно разговаривать с людьми на их «языке», об их «мире» и не заводить разговор в область конкретных решений или обсуждения технических проблем. В течение интервью желательно попросить человека описать по шагам процесс его работы. Все беспорядочные заметки, сделанные в процессе интервью, должны быть затем аккуратно структурированы в наборы требований и переданы представителю заинтересованной стороны для рецензирования. Интервью это интерактивный (диалоговый) процесс и очень важно, чтобы специалист, проводящий интервью, не выполнял свою работу поверхностно, а как можно чаще задавал вопрос «Почему?», докапываясь до сути (при этом не выступая в роли судьи, сразу оценивая, что важно, а что нет).

Существует множество способов сформулировать этот простой вопрос по-разному, например: «С какой целью вы делаете…?» или «Расскажите мне немного подробнее о…».
Несмотря на предварительную подготовку, интервьюер может не быть экспертом в предметной области и, следовательно, в ходе интервью всегда будут необходимы дальнейшие уточнения и подробные разъяснения. Поэтому не бойтесь задавать «глупые» (на первый взгляд) вопросы. Существует только единственный глупый вопрос - это тот, который не задан!!! Поэтому важно осознавать, что, в конечном итоге, именно пользователь (представитель заинтересованной стороны) несет ответственность за пользовательские требования – вы лишь помогаете им формулировать требования.

Не забывайте, что обсуждение сценариев использования с будущими пользователями системы является ключом к успеху.

Ниже мы приводим некоторые советы, которые помогут вам при проведении интервью:

- проводите интервью с представителями каждой категории заинтересованных сторон;
- обращайте серьезное внимание на мнение каждого из интервьюируемых;
- старайтесь документировать интервью и предлагайте собеседникам завизировать его (в некоторых случаях даже можно вести формальный протокол интервью и просить представителей заинтересованной стороны подписать его);
- выявляйте те сценарии, которые являются наиболее значимыми для данных собеседников, и «проговаривайте» с ними каждый шаг, выявляя, что же собеседник должен иметь возможность делать в каждом из состояний сценария (требования типа «возможность»);
- при необходимости создавайте новые сценарии использования, если они выявляются в ходе интервью, а затем формируйте пользовательские требования из этих новых сценариев;
- пытайтесь выяснить степень важности для интервьюируемого каждого из требований;
- если собеседник не может четко сформулировать какое-либо требование, то помогите ему - спросите о конечной цели этого требования, попросите проиллюстрировать предлагаемое требование на примере;
- выявляйте любые ограничения, которые могут быть известны собеседникам;
- давайте понять собеседникам, что именно их требования формируют систему;
- стимулируйте и «провоцируйте» собеседника активно включаться в обсуждение;
- никогда не высказывайте своего мнения относительно пользовательских требований;
- не будьте поверхностными во время интервью, старайтесь докопаться до сути;
- примите за правило такую последовательность действий – сразу же фиксируйте требование – как бы оно не звучало, - а потом постепенно, шаг за шагом его уточняйте.

Лучше всего обсуждение строить на принципе «от общего к частному». Очень важно быть уверенным в том, что все, что намечалось обсудить, было проговорено, и что при этом четко выделены области, не относящееся к делу.
Опыт проведения интервью подсказывает, что конкретные формы и содержание вопросов должны всегда зависеть от собеседников и ситуации.

Получение требований из неформальных документов

Неформальные документы, к которым могут относиться результаты исследований, служебные распоряжения, переписка, и т.д. могут также содержать требования, которые просто скрыты от первого взгляда. Задача и состоит в том, чтобы эти требования оттуда извлечь и сделать их «видимыми».
Однако, решая эту задачу, необходимо всегда фиксировать первичный источник, из которого они получены.
Более того, соответствующая заинтересованная сторона должна завизировать эти требования и «взять на себя ответственность» за них (стать их «владельцем»).

Получение требований из сценариев использования

После того, как был создан объединенный сценарий, становится возможным извлекать пользовательские требования непосредственно из него.
В некоторых случаях это делается простым перефразированием состояний (или целей) сценария использования. Например, состояние *готовность к отплытию* можно перефразировать и сформулировать «возможность» следующим образом: *пользователь должен иметь возможность подготовить парусную лодку к отплытию*.
В других случаях для этого требуется гораздо больше усилий.
Рис. 5.10 поможет нам продемонстрировать вышесказанное.

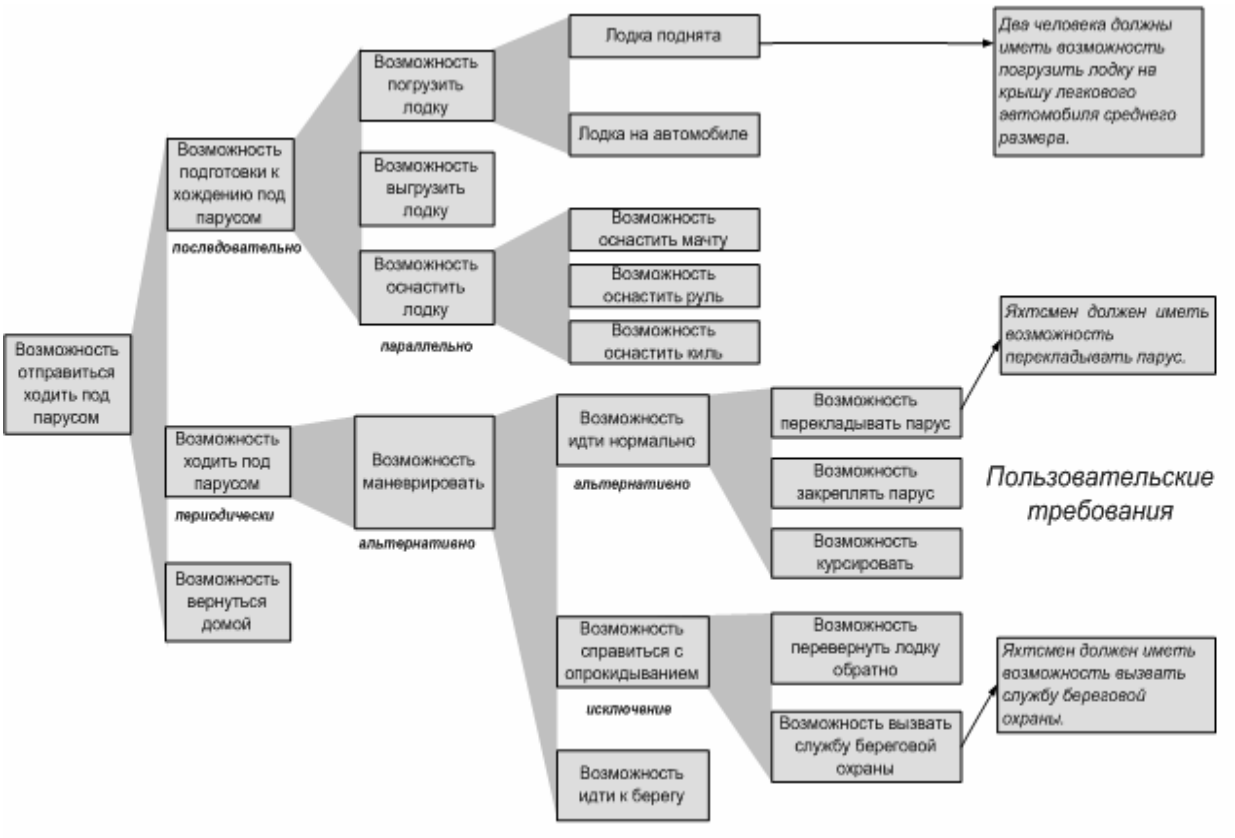


Рис. 5.10 Получение из сценария пользовательских требований типа «возможности».

Давайте рассмотрим следующее требование:

> *Два человека должны иметь возможность погрузить лодку на крышу легкового автомобиля среднего размера.*

Такая формулировка вызывает, как минимум, следующие вопросы:

- Насколько сильными должны быть люди?
- Что такое «легковой автомобиль среднего размера»?

Эти моменты должны быть в последствии обязательно прояснены. Однако сейчас – и это очень важно - даже такое требование должно быть зафиксировано. Совершенно неважно пока, что формулировка требования далека от совершенства и содержит неясные моменты. Все эти нечеткости формулировки будут устранены позднее. Самое важное - не упустить идею!
Перефразируя известное выражение, можно подвести черту под этим рассуждениями:

> *Если работа заслуживает того, чтобы ее делали,*
> *то она заслуживает того, чтобы ее делали хорошо!*

Более подробную информацию о том, как правильно формулировать требования можно найти в главе 4.

Семинар по сбору требований (requirements workshop)

Одним из путей сбора пользовательских требований является проведение семинаров.
Это достаточно эффективный способ быстрого сбора требований. Очень важно собрать заинтересованные стороны в благоприятной обстановке и добиться понимания ими того, что семинар по сбору требований не будет тяжелым и не займет у них много времени. Необходимо придерживаться определенной структуры проведения семинара, но, тем не менее, семинар должен носить характер свободного диспута.
Как показано на рис. 5.11, участники семинара должны быть подготовлены к семинару и осознавать, что от них требуется.
Например, они должны понимать суть концепций:

- заинтересованная сторона;
- сценарий использования;
- требование типа «возможность».

В зависимости от того, на каком этапе работ над проектом проводиться такой семинар, вполне вероятно, что уже существует предварительная версия набора требований, над которой вы будете работать. В противном случае, необходимо начать с того, что разделить всех участников на группы и попросить каждую группу разработать для будущей системы собственные сценарии использования. Затем необходимо попытаться объединить эти сценарии (по возможности) в один. Проверку отдельных и объединенного сценария желательно осуществлять уже при полном составе участников семинара. После обсуждения

сценариев и внесения в них необходимых изменений можно уже переходить к извлечению требований из сценариев.

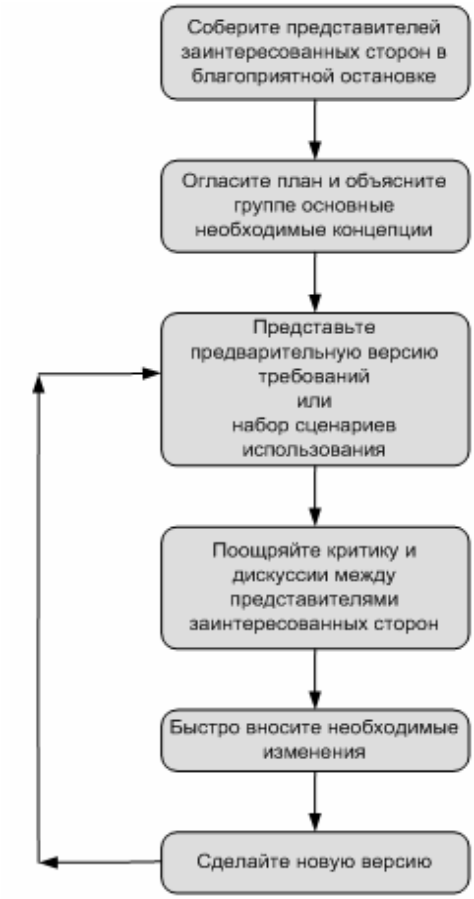


Рис. 5.11 Процесс получения пользовательских требований.

Как только будет получен предварительный набор требований, его нужно тут же выносить на обсуждение всех участников семинара. Необходимо поощрять критику и дискуссии. Возможность общения между различными группами заинтересованных сторон привносит весьма ощутимую пользу в процессе формирования требований. Часто случается, что эти люди, даже работая в одной компании, впервые видят друг друга на вашем семинаре. Поэтому и вам и им самим будет всегда интересно наблюдать, как взаимодействие между различными группами порождает требования, которые находятся на стыке интересов, - как желание иметь возможность делать что-то для одной группы пересекается с соответствующими желаниями и потребностями других групп.

При наличии проектора анализ и рецензирование полученных требований можно осуществлять в режиме on-line с привлечением всех участников семинара. Однако процесс идет более продуктивно, если весь коллектив собравшихся разделить на небольшие подгруппы и дать каждой из них собственный набор требований для независимого рецензирования и доработки, а затем уточненный полный набор требований просматривать еще раз уже всем коллективам. При таком подходе к организации семинара работа над требованиями к типичному проекту может продолжаться 3 – 4 дня.

Ключевыми элементами семинара являются формирование позитивного рабочего настроя, достижение нужного темпа работы и последующее поддержание его на протяжении всего семинара.

Следует заметить, что организация такого семинара потребует определенных усилий, однако полученный результат перекроет все затраты на его проведение.

Очень важно также, чтобы представители всех категорий заинтересованных сторон, присутствующих на семинаре, были уполномочены принимать решения.

Требования из опыта

Проблемы, обнаруженные непосредственными пользователями системы, - это просто «золотая жила» для дальнейшей работы, - однако очень часто эта информация даже не принимается к рассмотрению. Складывается некое негативное отношение к информации такого рода, поскольку она ассоциируется с проблемами, однако эта информация может принести очень большую пользу.

Несомненно, что чем раньше обнаружена проблема, тем дешевле стоит ее устранение – иногда для этого достаточно просто изменить формулировку требования или требований. Но с другой стороны, - если позволить легко, бесконтрольно и постоянно вносить изменения в проект, любой проект будет «похоронен» в самом начале. Поэтому если разработка системы предусматривает ее пошаговую реализацию (функционал внедряется не весь сразу, а постепенно), то становится возможным отложить внесение изменений до следующих версий.

Требования из прототипов

Если вы разрабатываете совершенно новую систему, аналогов которой не существует, в этом случае прототипы, макеты просто незаменимы. Прототипы дают представление о том, как система будет выглядеть. Так в области разработки программного обеспечения прототипы очень часто применяются для моделирования пользовательского интерфейса, поскольку будущим пользователям системы без наглядного примера бывает очень трудно представить себе как будет выглядеть этот интерфейс.

Главная проблема при разработке прототипов это то, что разработчики очень часто увлекаются и тратят на разработку макета очень много времени и сил. Разработка прототипов должна каждый раз позиционироваться как выделенный подпроект со своими собственными требованиями. Цель разработки прототипа должна быть четко определена, поскольку она нужна не только для разработки прототипа как такового, но она может оказать неоценимую помощь в формировании самих пользовательских требований.

Можно отметить три проблемы, которые чаще всего встречаются при разработке прототипов:

- разработчики обычно уходят в сторону и углубляются в детали;
- прототипы провоцируют заинтересованные стороны отклоняться в сторону конкретных решений;
- представители заинтересованных сторон могут быть настолько впечатлены прототипом, что могут потребовать его немедленного внедрения в промышленную эксплуатацию.

Избежать двух первых проблем вполне возможно с помощью хорошо сформулированных требований для прототипа. Для того же, чтобы исключить третью проблему, необходимо постараться раскрыть иллюзорную сущность такого прототипа, убедить заинтересованные стороны в том, что то, что они видят, является всего лишь макетом, который не может полноценно эксплуатироваться. При этом желательно использовать для прототипа средства и материалы, которые дают наглядное представление о временной сущности прототипа.

Ограничения в пользовательских требованиях

Ограничение это такой тип требования, который не добавляет возможности системе. Вместо этого, такой тип накладывает определенные обязательства на то, каким образом может быть достигнуто наличие той или иной возможности в системе.
Возьмем для примера следующее утверждение:

Клиент должен быть обслужен в течение 15 минут после размещения заказа.

Данное требование не меняет возможности системы, а всего лишь определяет характеристики предоставляемого сервиса.

Тем не менее, работая с ограничениями, необходимо соблюдать определенную осторожность. Совокупность ограничений, каждое из которых будет вполне оправдано, может сделать разработку в целом невозможной. Поэтому после анализа правомерности каждого конкретного ограничения необходимо провести анализ всей совокупности собранных ограничений, как единого целого.

И даже более - после того, как разработаны спецификации конкретных решений (определен дизайн), каждое ограничение должно быть проанализировано с точки зрения его влияния на стоимостные характеристики системы. В некоторых случаях наличие определенного ограничения может привести к необходимости введения дополнительных системных функций, что может увеличивать стоимость системы (например, реализация функции обеспечения безопасности, аварийной сигнализации или резервного копирования). До разработки спецификаций конкретных решений, можно только лишь догадываться о стоимости реализации ограничения. К сожалению, эта стоимость достаточно сильно зависит именно от выбранного варианта решения, но, даже несмотря на этот факт, предварительная оценка может быть сделана уже и сейчас – слишком большое количество ограничений может разрушить ваш проект (в том числе и с финансовой точки зрения).

Принято считать, что если ограничение накладывается на требование верхнего уровня, то действие ограничения наследуется также и всеми дочерними требованиями (нижнего уровня). Поэтому ограничение, по возможности, должна накладываться на как можно более высокий уровень иерархии возможностей, чтобы сократить применимость этого ограничения на каждом шаге, а, следовательно, и уменьшить его стоимость (см. рис. 5.12). Если же ограничение касается только одного требования, то оно может быть описано даже как часть самого требования.

Интересно отметить разницу между ограничениями на уровне пользовательских и системных требований. Ограничения на уровне пользовательских требований скорее относятся к результату, который должен быть получен. Системные же ограничения носят в большей степени «профессиональный» (более технический) характер и затрагивают скорее качество конечного продукта. Все пользовательские ограничения должны быть отражены в системных ограничениях. Иногда для этого необходимо изменить формулировку

ограничения, а иногда и делать ничего не нужно - пользовательское ограничение без изменений переходит в системное.

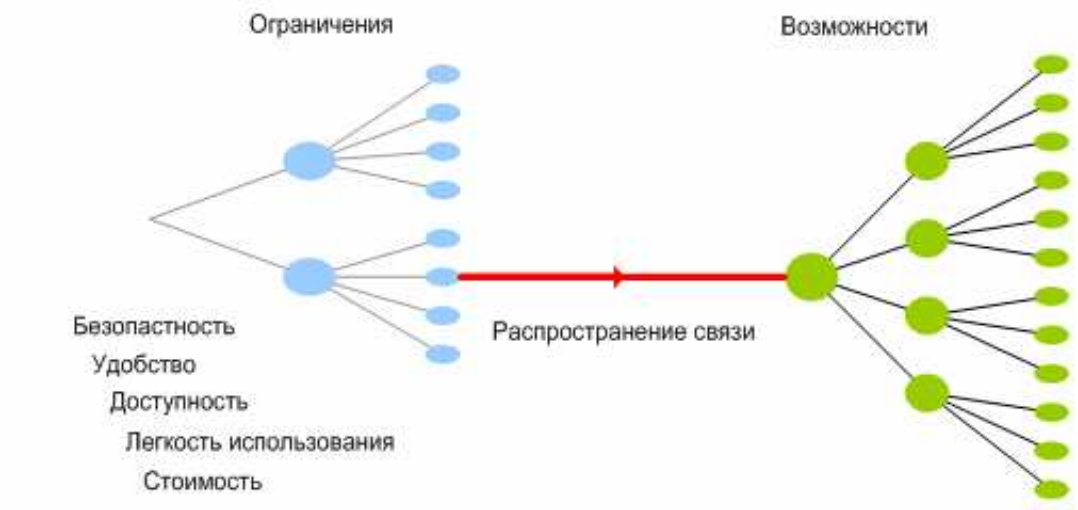


Рис. 5.12 Возможности и ограничения.

Уточнение требований

Анализируйте каждое требование с точки зрения его контекста и каждый раз убеждайтесь в том, что:

- требование находиться на своем месте;
- требование удовлетворяет качественным критериям, которые описывались в главе 4.

Определение стратегии проверки

Формирование стратегии проверки проходит по двум параллельным направлениям (см. рис. 5.13). Эта тема обсуждается подробнее в последующих разделах.

5.5.3 Определение критериев приемки

Понимание того, как заинтересованные стороны собираются удостовериться в том, что их требования удовлетворены, является очень важным аспектом разработки требований.
Ответ на вопрос:

Что убедит вас в том, что данное требование удовлетворено?

может помочь даже более точно сформулировать само требование. Вот почему этот вопрос очень часто используется во время интервью представителей заинтересованных сторон.

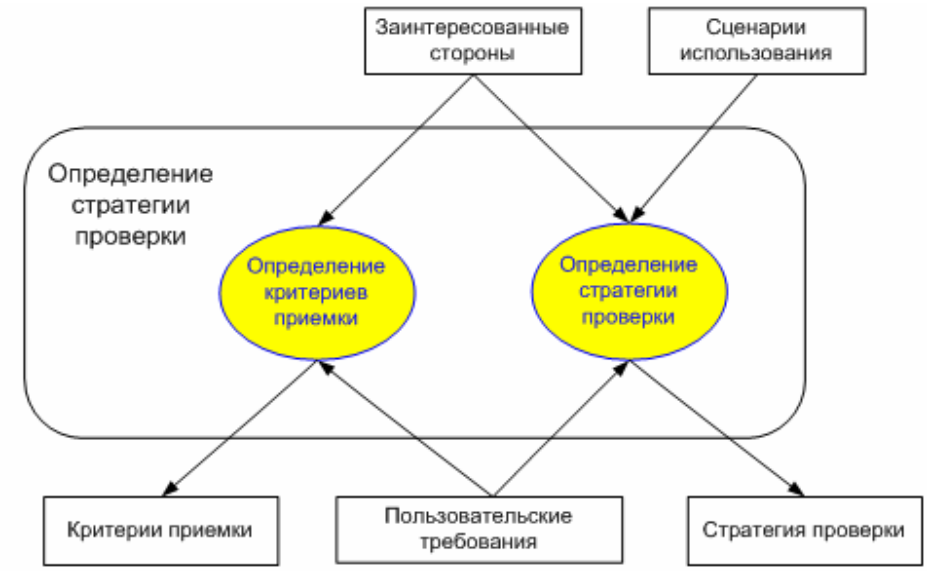


Рис. 5.13 Процессы определения стратегии проверки.

Обычно на этот вопрос отвечают двумя разными способами:

- в качестве примера может быть описана функциональная ситуация, в которой может быть продемонстрировано реализация требования, и/или
- может быть зафиксировано некое числовое значение, достижение которого должно быть продемонстрировано системой.

Первый способ ответа на вопрос ведет непосредственно к процессу создания наборов тестов, испытаний и демонстраций возможностей, которые могут являться как раз частью стратегии проверки.
Во втором случае мы получаем значения показателей, которые должны быть достигнуты в результате успешного прохождения тестов или испытаний. Другими словами, мы формулируем критерий приемки требования или отметку о выполнении теста («пройден» \ «не пройден»).

Критерий приемки определяет - для каждого требования – показатель, достижение которого будет характеризовать «успешное» завершение теста. Критерий приемки обычно записывается в виде значения атрибута, ассоциированного с данным требованием. Другими словами, в большинстве случаев между требованием и критерием его приемки должна существовать связь типа «один к одному».
В случае с рестораном можно предположить, например, что критерий приемки, оценивающий деятельность ресторана, скорей всего будет иметь значение «успешно». Успех может измеряться по целому ряду показателей:

- доходность ресторана;
- показатель возврата инвестиций;
- объем информации о ресторане в путеводителях, газетах, местоположение в рейтингах и т.п., как отражение положительной репутации ресторана;

- насколько вперед требуется зарезервировать место в ресторан, чтобы в него попасть (т.е. на какое время вперед ресторан уже полностью зарезервирован).

При этом разные заинтересованные стороны могут иметь различные мнения по поводу того, что является мерилом успеха. Например, управляющий банком, клиентом которого является владелец ресторана, будет больше заинтересован в двух первых показателях, а шеф-повар ресторана будет скорее заинтересован в двух последних показателях.

Следовательно, очень важно для каждого требования определить показатели критериев приемки, формулируемые каждой из заинтересованных сторон.

5.5.4 Определение стратегии проверки

Способы, с помощью которых осуществляется приемка системы, зависят от характера системы.

Для больших систем имеющих только одно внедрение, например, таких как система управления воздушным транспортом страны, стратегия проверки должна строиться на том, что необходимо убедиться, что вся требуемая функциональность реализована; что авиадиспетчеры вполне довольны простотой использования системы; что система имеет достаточное быстродействие в периоды высокой нагрузки и т.д.

Это может потребовать проведения многосторонних тестов и испытаний. В первую очередь необходимо продемонстрировать все возможности системы при невысокой нагрузке. В том случае, если обнаружатся проблемы при невысокой нагрузке, то уже не будет никакого смысла переходить к дорогостоящим испытаниям при более высокой нагрузке.

Необходимо всегда учитывать стоимость конкретно разрабатываемой стратегии проверки.

Организация всесторонних интенсивных испытаний всегда очень дорогое удовольствие, поэтому к выбору типа испытаний нужно подходить очень аккуратно и здесь лучше все-таки следовать принципу «последовательности», так, например, прежде чем приступать к испытаниям корабля в открытом, да еще и в штормовом море, лучше вначале посмотреть на его поведение в спокойной бухте.

Немаловажно также не упускать из виду и общую стоимость всех совокупных проверок и испытаний. Суммарная стоимость проверок всегда должна быть соотнесена со стоимостью последствий от «необнаружения» недостатков в готовом продукте.

Так, например, если эксплуатация системы связана с риском для жизни людей, возможностью нанести непоправимый вред окружающей среде, или большими финансовыми потерями, то стратегия проверки должна быть очень подробной и выверенной (пусть и дорогой), чтобы иметь возможность убедиться в высочайшем качестве и безопасности системы. В случае же, когда риски и потери, связанные с ненахождением дефектов в конечном продукте, относительно невелики, то, может быть, стоит подумать о том, как разработать более дешевую стратегию проверки.

Практический вывод, вытекающий из формирования стратегии, следующий.

Если требование нельзя проверить тем или иным путем, то это требование не может называться требованием. Правильно разработанное требование это такое требование, которое легко понять и выполнение которого может быть легко продемонстрировано.

5.6 Заключение

Пользовательские требования должны быть краткими и понятными насколько это возможно.
Пользовательские требования не должны быть техническими, но в то же время они должны быть реалистическими (реализуемыми).
В пользовательских требованиях больше внимания должно быть уделено ролям и ответственности, которые должны быть правильно распределены между категориями заинтересованных сторон.

При разработке пользовательских требований наиболее часто встречаются следующие проблемы:

- чрезмерный уклон в сторону решений;
- недостаточное внимание к идентификации реальных проблем, которые нужно решить;
- отсутствие у заинтересованных сторон понимания того, что именно они должны нести ответственность за требования (владеть требованиями).

Пользовательские требования следует разрабатывать как можно раньше, потому что именно они определяют возможности, необходимые заинтересованным сторонам, при этом выраженные удобным и понятным для них языком.
Таким образом, необходимо сосредоточивать усилия именно на проблемной области, а не на области решений.
Пользовательские требования должны быть хорошо структурированы и (там, где возможно) должны быть «увязаны» с первоисточником информации, из которого они были получены.
Необходимо четко определить границы ответственности сторон при разработке пользовательских требований. Заинтересованные стороны должны нести ответственность за содержание требований. Требования должны быть ограничены рамками бюджета, соблюдение которого должно контролироваться финансистами. За сбор и оформление требований должны отвечать обученные специалисты-аналитики.

6 Разработка требований в области решений

*Никогда не говорите людям, **как** им выполнить их работу. Скажите им, **что** нужно сделать, и они удивят вас своей находчивостью.*

Джордж Смит Паттон (George Smith Patton),
генерал, 1885–1945

6.1 Что такое область решений?

Область решений, эта как раз та область, в которой инженеры задействуют все свои способности и изобретательность, пытаясь решить обозначенные проблемы. Основное отличие области решений от области проблем состоит, безусловно, в том, что разработка требований в области решений начинается с набора четко определенных требований, а разработка требований в области проблем стартует с нечеткой цели или размытого списка общих пожеланий. Конечно же, оценка, - насколько «хорошо сформулированы» требования, поступающие на вход области решений, - зависит от квалификации и аккуратности специалистов организации заказчика, которые создавали этот список (в области проблем). Самый идеальный случай – все входящие требования хорошо сформулированы и каждое из них может быть протестировано.

Как отмечалось в главе 2, решение достаточно редко выполняется за один подход (см. рис. 6.1).

На каждом этапе разработки вначале проводится моделирование и анализ, что, во-первых, улучшает понимание входящих требований, а, во-вторых, обеспечивает надежную базу для последующей разработки производных требований для следующих уровней (этапов). Количество уровней зависят от характера и природы создаваемой системы, а также от степени новизны предлагаемых проектных решений. И совершенно неважно, какое количество уровней вам придется создать, - значение имеет лишь степень детализации решений на каждом из уровней.

На каждом из уровней разработки требований в области решений инженеры вынуждены принимать мини-решения, которые шаг за шагом продвигают их вперед к общей намеченной цели. Каждое из таких мини-решений, по сути своей природы, завоевывает собственное жизненное пространство на поле общего дизайна, т.е. каждое из решений ограничивает возможности других. Но общее развитие прогресса невозможно при отсутствии таких решений (шагов). А поскольку некоторые инженеры и разработчики зачастую сразу же начинают вдаваться в излишнюю детализацию даже на самых ранних этапах (т.е. захватывать большие куски пространства на поле общего дизайна), то надо постараться исключить такую тенденцию или, как минимум, всячески ее ограничивать с той целью, чтобы дать возможность работать всей команде, чтобы использовать способности и таланты других специалистов, чтобы стимулировать творческий подход к совместному решению проблем на всех этапах, и, в итоге, получить по-настоящему инновационное решение. Ведь ясно же, что такое инновационное решение не может быть получено, если на самых первых шагах проекта создавать ограничения (жизненного

пространства), связанные с принятием преждевременных излишне детализированных решений.

Первым этапом в системных разработках в области решений обычно является преобразование пользовательских требований в системные. Системные требования как раз и будут определять, что именно система должна выполнять для того, чтобы решать проблемы, поставленные в пользовательских требованиях. На рис. 6.1 схематически проиллюстрировано вышесказанное – приведена ориентировочная последовательность шагов (сверху вниз) универсального процесса разработки системных требований.

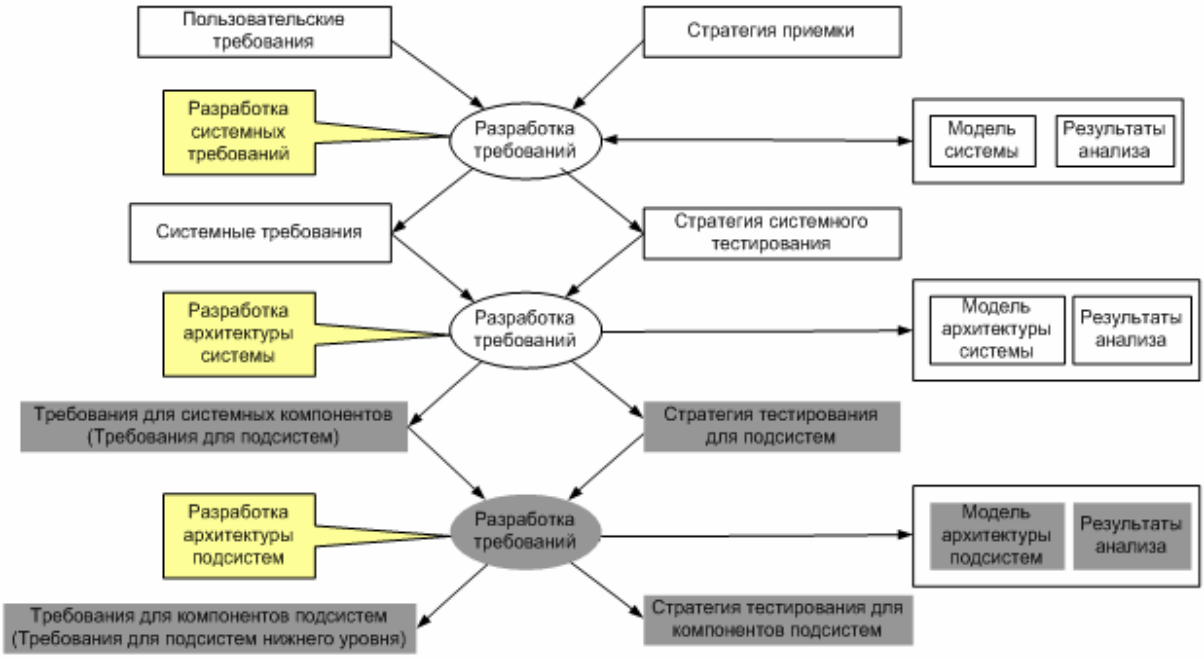


Рис. 6.1 Возможная последовательность шагов процесса разработки системных требований.

Появление излишней детализации проектных решений особенно проблематично на первом шаге. Во избежание этого, модель системы на этом этапе (см. рис.6.1) должна быть достаточно абстрактной, позволяя лишь определять необходимую системе функциональность, но не углубляться в ее излишнюю детализацию.

Следующим шагом разработки системных требований является создание архитектуры системы. На рис. 6.1 это обозначено в виде второго (сверху) уровня основного процесса. Архитектура должна быть представлена в виде набора компонентов системы, которые, взаимодействуя, и порождают системные свойства, определенные в системных требованиях.

На следующем шаге производные требования, вытекающие из процесса разработки архитектурного дизайна (см. нижний уровень на рис. 6.1), определяют те требования, которым должны удовлетворять компоненты системы.

Такой процесс разработки требований должен проходить с постепенным углублением в детали реализации.

В данной главе мы уделим особое внимание процессу преобразования пользовательских требований в системные, поскольку это самый сложный этап для большинства проектов.

Именно на этом этапе, как показывает практика, преждевременные решения появляются слишком быстро.

6.2 Получение системных требований из пользовательских

На рис. 6.2 приведен пример универсальной схемы преобразования пользовательских требований в системные.

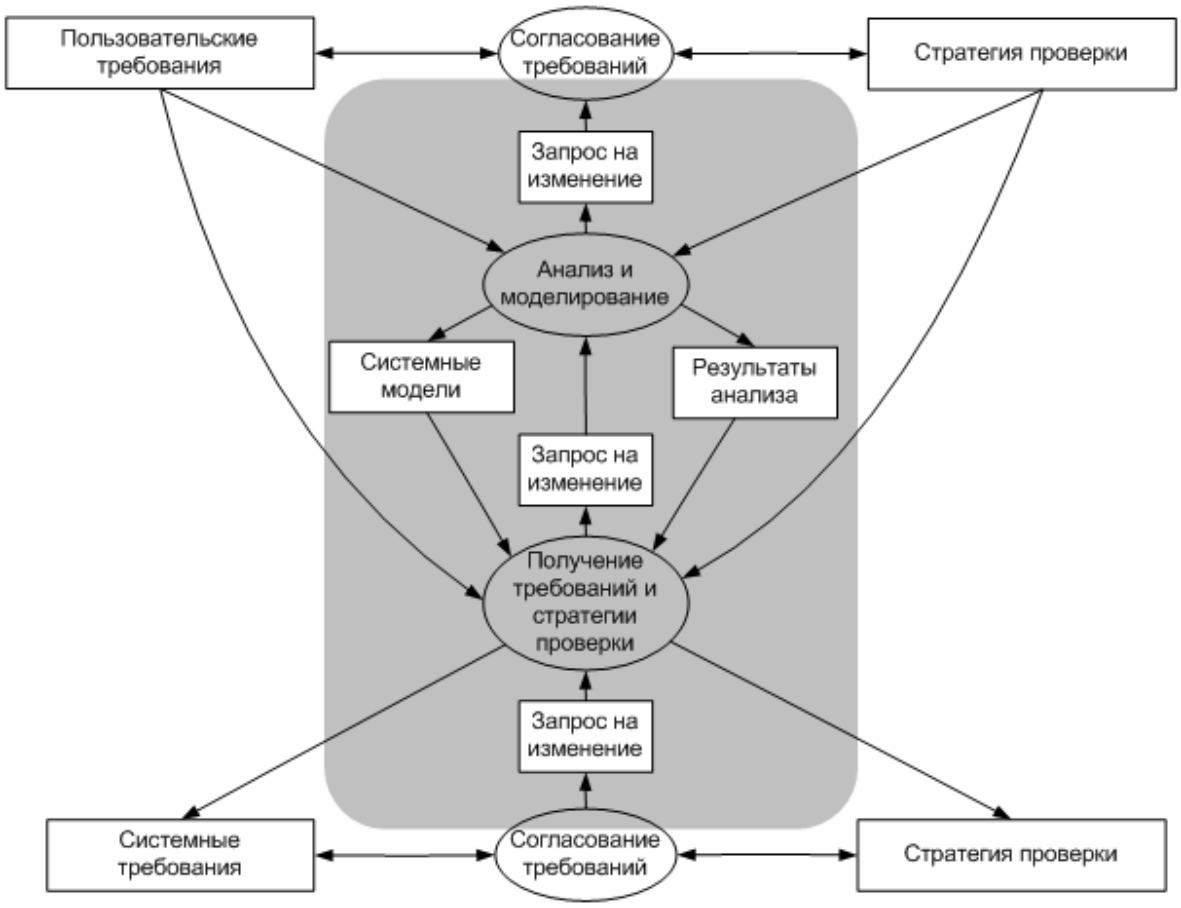


Рис. 6.2 Универсальная схема процесса для получения системных требований.

Как и на любом другом этапе, процесс начинается с согласования входящих требований, которыми, в данном примере, являются пользовательские требования.
Но, согласовывая входящие требования, вы должны отдавать себе отчет в том, что те люди, которые разрабатывали пользовательские требования, отнюдь не следовали рекомендациям, изложенным ранее в этой книге, а может даже и не читали ее вовсе. Поэтому напротив, необходимо достаточно серьезно и строго подойти к оценке пользовательских требований и связанной с ними стратегии проверки на соответствие критериям «правильности» требований (о чем вы уже прекрасно осведомлены, если добрались до этой главы).

6.2.1 Разработка системной модели

Во избежание опасной тенденции углубиться во множественные детали реализации, инженеры всегда должны работать в рамках контекста конкретной модели (см. рис. 6.1), уже достаточно детализированной, чтобы получать требования, которые следует формулировать, исходя из того, **что** система должна делать, а не то, **как** она должна это выполнять. Степень детализации вытекающих требований, конечно, зависит от этапа (уровня) разработки, на котором в данный момент находятся инженеры, однако, чтобы не ошибиться, лучше всегда руководствоваться простым правилом «не вносить подробностей больше, чем это необходимо».

Тенденция «сваливания» в преждевременное углубление в детали реализации наиболее ярко проявляется на начальном этапе (верхнем уровне) разработки, когда пользовательские требования, сформулированные в терминах области проблем, могут сразу переводиться в форму системных требований высокого уровня, обозначающих, что система должна делать, чтобы решить проблемы, сформулированные заинтересованными сторонами.

Эти проблемы возникают потому, что достаточно сложно (но надо!), работая, удержаться на определенном уровне абстракции. Но это характерное явление – создание абстрактной системной модели, из которой потом рождаются системные требования, всегда происходит в муках.

На любых уровнях ниже такие проблемы возникают реже. Разработка тут проводиться уже в контексте конкретных архитектурных решений, где проектировщики и разработчики чувствуют себя достаточно комфортно, поскольку вовлекаются в процесс определения, *как* именно система будет работать. Но даже на любом из этих уровней необходимо уделять должное внимание тому, чтобы степень детализации соответствовала именно этому уровню разработки. Соответственно, и в отношении архитектурных моделей, представленных в виде набора компонентов системы, функционирующих вместе, необходимо внимательно следить за тем, чтобы эти компоненты определялись в терминах того, какие функции они должны выполнять, без подробностей относительно того, каким образом они должны это делать. Иными словами, необходимо следить за тем, чтобы компоненты представляли собой «черные ящики», выполняющие сформулированные в требованиях задачи, при отсутствии описания их внутренней реализации.

Следующие разделы этой главы посвящены методам разработки системных моделей для получения на их основе системных требований. В развитие этой темы, описывается, как данный подход может использоваться на более низких (более детализированных) уровнях разработки требований.

6.2.2 Разработка системных моделей для получения системных требований

Системная модель должна разрабатываться с определенным уровнем абстрактности, который подразумевает описание:

- внутренней функциональности, которую система должна реализовывать. Для того чтобы избежать преждевременных детальных решений, внутренняя функциональность

должна быть описана в терминах того, **что** система должна выполнять, а не **как** это делать;
- функциональности, необходимой для взаимодействия системы со своим окружением (с внешними системами);
- функциональности, позволяющей людям взаимодействовать с системой;
- функциональности, которая предохраняет систему от «ложного срабатывания», что может быть вызвано вредным (опасным) воздействием со стороны других систем из ее окружения. (Отметим, что это воздействие не всегда является *сознательно* угрожающим; например, электромагнитное излучение другого оборудования.) При этом «защитная» функциональность также должна «работать» и в обратном направлении - предотвращать вредное воздействие системы на окружающую среду.

На рис. 6.3 проиллюстрировано взаимодействие перечисленных функциональностей между собой и с элементами внешнего окружения.

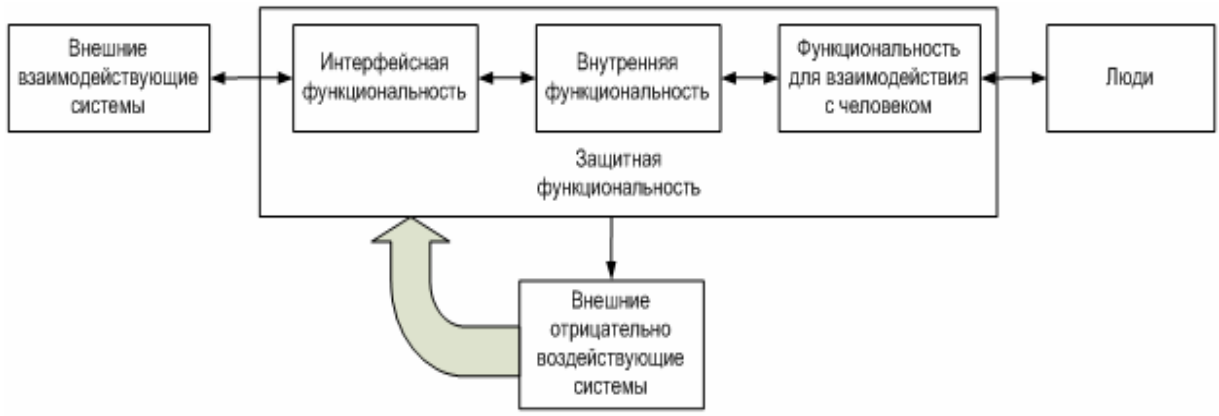


Рис. 6.3 Контекст системы и типы функциональности.

Очевидно, что состояние системы по отношению к ее окружению должно характеризоваться:

- существующими системами, с которыми разрабатываемая система должна взаимодействовать;
- типами пользователей, которые будут взаимодействовать с системой;
- опасными событиями, от которых система должна быть защищена;
- негативными последствиями, которые должны быть предотвращены.

Функциональность может быть смоделирована несколькими различными способами:

- операции или методы классов на диаграммах классов;
- диаграммы сообщений (MSC);
- диаграммы переходов состояний;
- блок-диаграммы функциональных потоков;
- процессы на диаграммах потоков данных.

На практике приходится использовать несколько различных типов моделей для описания всех требуемых аспектов функциональности системы. А поскольку каждая модель содержит информацию определенного типа, а каждая технология моделирования использует свою собственную семантику, то информация в одной модели может быть «отделена» от информации в другой модели. Но с другой стороны, одна и та же информация может появляться в нескольких моделях. В последнем случае очень важно быть уверенным, что если информации меняется в одной из моделей, то это изменение отражается и во всех остальных моделях, в которых эта информация встречается. В идеальном варианте это может происходить автоматически с помощью сопряжения (интеграции) моделирующих инструментов. Если же это по каким-то причинам не может быть организовано, то необходимо проявлять исключительную педантичность и аккуратность, чтобы удостовериться в том, что любое изменение информации в одном месте *идентично* отражается в других связанных моделях. Диаграмма Венна (рис. 6.4) иллюстрирует, что некоторые модели могут быть представлены в виде отдельных островков информации, в то время как другие модели могут иметь общую информацию, возможно представленную в разных формах. Диаграмма также демонстрирует тот факт, что часть информации о системе не отражена ни в одной из моделей.

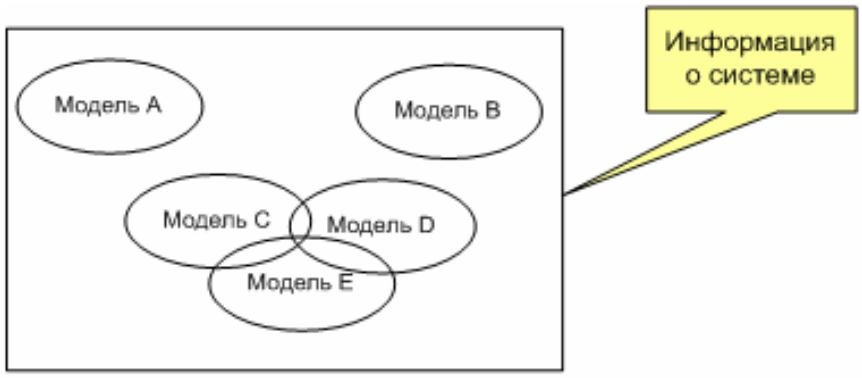


Рис. 6.4 Границы системных моделей.

Внутренняя функциональность

Внутренняя функциональность является исходным элементом для разработки системных требований, потому что определяет то, что система должна делать. Для того чтобы получить системные требования, необходимо разработать функциональную структуру или модель системы, которые будут являться основой для дальнейшей работы. Такая модель должна предполагать возможности декомпозиции системы и позволять выделение модулей или высокоуровневых компонентов, например, подсистем. Конечно, термины «*модуль*» или «*подсистема*» провоцируют проектировщиков и разработчиков мыслить больше в категориях реализации, нежели чем в направлении конкретизации и спецификации, поэтому употребление этих терминов является не очень хорошей практикой (особенно в области разработки программного обеспечения). В большинстве систем, «сползание» в область физических моделей не рассматривается, как некая чрезвычайная ситуация, поскольку сама по себе область применения систем может иметь физическую природу.

Как альтернатива этой терминологии, стимулирующей появление преждевременных решений, существует тенденция (можно даже сказать «мода») употреблять термин «*объект*», говоря о декомпозиции. Особенно это популярно в области разработки ПО, поскольку объект, в этом случае, может описывать реальный предмет из области проблем. Такой подход помогает предотвратить попытки мышления проектировщиков в терминах решений. В соответствии с этим подходом функциональность системы может быть представлена как совокупность методов и операций над объектами, а также в виде описания взаимодействия объектов между собой.

Использование такого объектно-ориентированного подхода позволяет также упростить задачу установления связей от системных требований к пользовательским, поскольку зачастую некоторые объекты проявляют тенденцию «просматриваться», как в области проблем, так и в области решений.

Помимо описания того, что система должна делать, от системной модели требуется также обозначить подразумеваемое поведение системы. Эта информация может быть представлена разными способами. Достаточно вспомнить, что модели в этой области обычно отражают тот факт, как несколько ролей (актеров) могут одновременно взаимодействовать друг с другом.

В качестве примера таких нотаций можно вспомнить диаграммы последовательности сообщений (Message Sequence Diagrams = MSC) и диаграммы поведения (Behaviour Diagrams). MSC-диаграммы на протяжении уже достаточно долгого времени и с большим успехом используются в телекоммуникационной отрасли. А диаграммы поведения впервые начали использоваться в 70х годах прошлого века в США при разработке системы BMEWS (Ballistic Missile Early Warning System) раннего оповещения о возможном ударе баллистических ракет, а позже были включены в средства моделирования RDD-100 (Ascent Logic Corporation) и CORE (Vitech Corporation).

Поведение можно моделирование также с помощью диаграмм переходов состояний (State Transition Diagrams) и диаграмм состояний (State Charts). Диаграммы переходов состояний имеют свои ограничения. С их помощью можно отобразить только последовательность состояний; причем так, что в каждый конкретный момент времени моделируемый объект может находиться только в одном из этих возможных состояний. Этот тип диаграмм не может отобразить иерархию состояний моделируемого объекта напрямую. Поэтому для того, чтобы смоделировать иерархию состояний приходится разрабатывать диаграммы переходов состояний для каждого уровня иерархии. В некоторых случаях это может привести к тому, что в некоторые моменты времени разные наборы диаграмм могут пребывать в активном состоянии. Такое сложное поведение бывает трудно осмыслить и понять. Для преодоления подобных трудностей рекомендуется использовать диаграммы состояний, поскольку они изначально разрабатывались для моделирования иерархий состояний в целом, а также для моделирования параллельных состояний.

Для любой системы необходимо также определить то, какую именно информацию и как необходимо обрабатывать и хранить.

В некоторых системах, например, в системах учета страховых компаний, необходимо, чтобы информация собиралась и хранилась в течение многих лет. В других системах, например, в системе обработки данных, полученных с помощью радара, для обеспечения управления воздушным движением, часть информации должна храниться достаточно долго, например, планы полетов, а другая информация, например, текущее положение самолета, не актуальна уже в следующую секунду, поскольку самолет переместился.

Следовательно, необходимо определить следующие параметры, касающиеся хранения информации, относящейся к разрабатываемой системе:

- срок жизни информации, *т.е. какое время эта информация является достоверной, и какое время она должна храниться*;
- частота обновления информации, *т.е. насколько часто информация должна обновляться (секунды, минуты или часы)*

Очень важно также знать и объем информации, который будет обрабатываться системой. Этот показатель может оказать существенное влияние на архитектуру системы.

Интерфейсная функциональность

Здесь необходимо определить характер будущего взаимодействия системы с любыми другими внешними системами.
Это взаимодействие может быть связано и с передачей информации, и с перемещением вещественного материала между системами. Эта передача может носить только однонаправленный характер или же «функционировать» в обоих направлениях; причем при наличии или отсутствии ограничений, налагаемых на такую передачу. В некоторых случаях необходимо даже организовать временное хранение передаваемых\принимаемых объемов информации или материалов (например, буфер, или склад), если они не могут быть обработаны сразу. Могут существовать и определенные требования относительно времени реакции системы на внешние воздействия со стороны других систем.

Природа самих интерфейсов также может существенно варьироваться. Однако всегда должен существовать некий базовый документ, который бы четко характеризовал, за что именно отвечает и что именно обеспечивает каждая часть (уровень) интерфейса, как составляющая общего целого. Эти обязательства обычно документируются в виде специального документа ICD (Interface Control Document), описывающего интерфейс. В тех случаях, где взаимодействия между системами обеспечиваются с соблюдением национальных или международных стандартов, такой стандарт автоматически становится документом ICD, руководствуясь которым могут функционировать все составляющие части. В тех случаях, когда интерфейс не так четко определен, требования к интерфейсу, все же, должны быть согласованы и зафиксированы в письменном виде. Контроль за выполнением требований такого рода практически всегда сопряжен с проблемами, поскольку не существует одной общепризнанной организации, которой делегировано право контроля за интерфейсами. Это зачастую приводит к тому, что каждая из сторон, использующих интерфейс, может иметь собственные версии такого ICD документа, или, что еще хуже, - на свое усмотрение интерпретировать один и тот же документ.

Обычно выполнение рекомендаций документов, описывающий интерфейс, контролируются той организацией, которая отвечает за разработку системы, предполагающей взаимодействие с другими системами. Однако в том случае, когда разрабатывается совершенно новая система, найти организацию, которая взяла бы на себя ответственность за контроль такого рода, является большой проблемой.
Очень часто бывает так, что для ранее разработанных систем документации, описывающей интерфейс нет или она очень плохая. Более того, возможно, что разработчик системы уже и не поддерживает ее или передал ее на обслуживание своему клиенту или другой организации. Это порождает свои трудности при определении интерфейсов.

Необходимо уделять большое внимание тому, чтобы у вас была полная уверенность, что все обязательства по соблюдению интерфейсов отражены в производных требованиях на всех соответствующих уровнях, и что четко, насколько это возможно, определена организация, которая будет осуществлять контроль за соблюдением интерфейса.

Функциональность, обеспечивающая взаимодействие с человеком

Основным вопросом в области общения системы с человеком является определение того, какое именно взаимодействие им может потребоваться. Здесь не следует забывать про условия, в которых человек взаимодействует с системой, поскольку они оказывают влияние на то, как именно человек будет работать с системой.
Например, человек, работающий в кабинетных условиях, находится в тепле и может работать с данной системой без перчаток; а вот другие люди вынуждены работать с этой системой в экстремальных условиях, скажем, при низких температурах или других вредных условиях, поэтому им может потребоваться защитная одежда. Тогда конструкция, например, дисплея и клавиатуры должна учитывать и эти факторы.

Защитная функциональность

Внешняя среда, в которой должна функционировать система, существенным образом влияет на требования по безопасности и защите системы.
Например, для банковских систем необходимо обеспечить защиту информации и финансов от доступа к ним неавторизированных лиц. В автомобиле необходимо гарантированно обеспечить остановку автомобиля при нажатии человеком на педаль тормоза.

Возможно, следует принимать во внимание и другие - особенно конкурирующие, - системы, которые функционируют в окружении разрабатываемой системы. Такого рода конкуренция может оказывать разное влияние на систему. Например, если ведется разработка системы для on-line предоставления клиентам банковских услуг, то присутствие или возможное появление на рынке конкурирующих банковских систем, может существенным образом влиять на сроки разработки вашей системы.

В других случаях, «конкуренция» может выражаться в непосредственном влиянии одной системы на другую, например, радиоизлучение одной системы, может перегружать чувствительные приемники другой системы. Примером этого является запрет на пользование мобильными телефонами на борту самолета, поскольку они могут отрицательно влиять на системы навигации самолета.

Системные транзакции

Разрабатывая систему, бывает очень полезно еще раз пересмотреть и проанализировать существующие сценарии использования, полученные от заинтересованных лиц, или, – если таковые отсутствуют, - самим создать соответствующие наборы сценариев. Тогда на системную модель вы сможете «смотреть» сквозь призму этих сценариев, убеждаясь, что они функционируют надлежащим образом в рамках создаваемой системы (см. рис. 6.3). Такая проверка осуществляется при помощи, так называемых, «системных транзакций». Этот подход позволит вам еще раз убедиться, что на этапе объектной или функциональной декомпозиции системы не был упущен ни один из элементов ее функциональности.

(Попутно заметим, что в данном случае значение термина «системная транзакция» отличается от его значения, приведенного при упоминании метода CORE в главе 3).

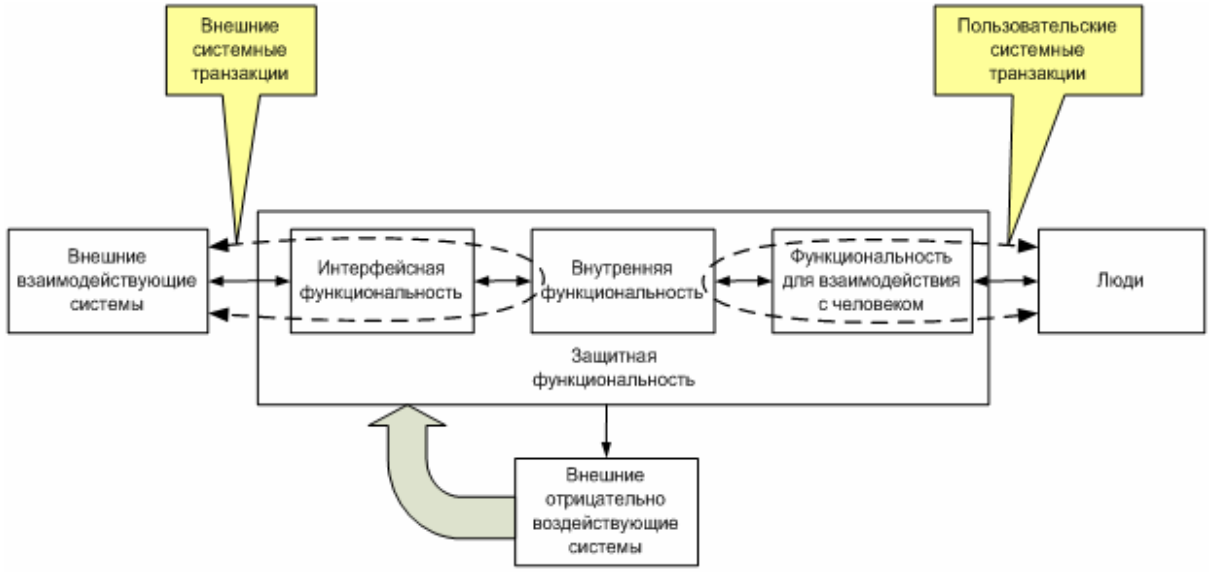


Рис. 6.5 Системные транзакции.

Системные транзакции, которые на рис. 6.5 помечены, как «*пользовательские системные транзакции*», получены из сценариев использования. Вместе с тем, рисунок демонстрирует, что могут быть и другие транзакции, являющиеся производными от способа, которым разрабатываемая система должна взаимодействовать с внешними системами, - «*внешние системные транзакции*».

Использование системных транзакций поощряет проектировщиков, как бы, приподняться над системой и окинуть ее «глобальным» взглядом, потому что всегда существует опасность «за деревьями забыть о лесе». Другими словами, концентрируясь на работе с мелкими деталями, можно забыть про всю картину в целом, т.е. упустить из виду, как отдельные части системы, взаимодействуя друг с другом, должны обеспечивать достижение глобальной цели.

Режимы функционирования системы

При разных обстоятельствах от системы может требоваться различная функциональность (поведение).

Типичным примером, поясняющим вышесказанное, может служить режим, предназначенный для обучения пользователей информационных систем, при котором система должна – даже при самых нерадивых студентах - обеспечивать целостность промышленных данных. В качестве другого примера можно привести аварийный режим работы системы, когда она функционирует по особому сценарию, или – как это обеспечивают военные системы - в зависимости от уровня тревоги могут быть предусмотрены различные режимы функционирования.

Все это, так или иначе, может быть связано со специальными сценариями использования, которые диктует заинтересованная сторона.

Дополнительные ограничения

В дополнение к уже упомянутым, существуют и другие ограничения, которые необходимо учитывать. Возможно наиболее важные из них - это те, которые обеспечивают безопасность применения системы и ее готовность к сертификации. Для обеспечения этих условий и разрабатываются дополнительные требования, которые, несомненно, могут оказывать значительное влияние не только на саму разработку системы, но и даже на выбор тех средств, с помощью которых она будет разрабатываться. Представители уполномоченных органов должны иметь возможность убедиться в том, что разработанная система удовлетворяет действующим нормам и безопасна для установки и использования. Каждому самолету, например, выдается специальный сертификат, подтверждающий его пригодность к полетам.

Другой набор ограничений может быть обусловлен возможностями производства системы. Например, возможно существуют определенные производственные мощности, которые могут быть использованы для производства системы, или проект системы должен быть таким, чтобы максимально снизить затраты на производство.

6.2.3 Абстрактная модель банка

Приводя пример банковской системы, авторы хотели бы, в первую очередь, отразить модель информационных потоков, хотя совершенно ясно, что существует масса и других аспектов поведения системы, которые могут быть описаны схожими моделями.

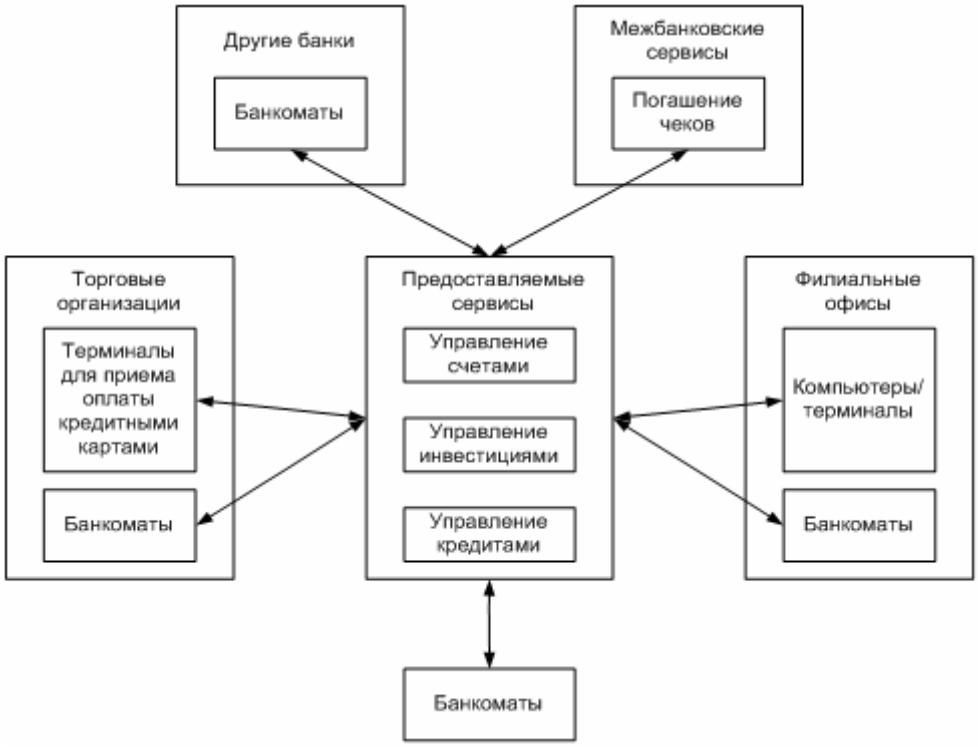


Рис. 6.6 Абстрактная модель банка.

В конечном итоге, желательно иметь несколько разных системных моделей, каждая из которых отражают различные аспекты функционирования, например, информационные модели, модели потоков данных, модели обеспечения защиты информации.

На рис. 6.6 изображена абстрактная модель банка, описывающая различные типы мест размещения оборудования, из которых могут быть инициированы транзакции.

Внутренняя функциональность

Естественно, что основная внутренняя функциональность банковской системы связана с предоставлением банковских услуг своим клиентам: текущий расчетный счет, сберегательный счет, кредиты, инвестиционные портфели. Для обеспечения этих услуг система должна иметь возможность собирать, обрабатывать и хранить информацию. Здесь очень важно дифференцировать типы (или классы) хранимой информации (напр., *счета, клиенты*); связи, существующие между ними (напр., *сколько счетов в банке может иметь клиент*); а также дополнительные характеристики для каждого типа хранимой информации (напр., *время хранения*, *частота обновления*, *объем*).

Необходимо также определить и то, *каким образом* эта информация собирается, распределяется и обрабатывается.

Следующей важной характеристикой банковской системы является количество и дислокация источников информации и/или источников транзакций. Этот список может включать офисы и филиалы банка, банкоматы, карточные терминалы в торговых точках и т.д.

С точки зрения производительности, необходимо определить вероятную нагрузку на систему, выраженную в количестве транзакций определенных (и смешанных) типов в единицу времени, с которой система должна справляться. Разумеется, в реальной жизни эта нагрузка не будет равномерной, а будет сильно варьироваться в зависимости от времени суток, дня недели и месяца года. Здесь не следует забывать об ограничениях, которые могут быть связаны с пропускной способностью существующих линий и каналов связи, а также других средств (оборудования) коммуникации.

Интерфейсная функциональность

Основными интерфейсами для банковских систем такого типа являются интерфейсы для взаимодействия с системами других банков, обеспечивающими перевод денежных средств, и интерфейсы для взаимодействия с их банкоматами.

При этом также необходимо иметь в виду, что несколько банков могут использовать и другие интерфейсы, поскольку могут быть объединены в собственную систему для обслуживания, например, банковских чеков или аналогичных платежных средств.

С большой долей вероятности можно предположить и наличие другого типа интерфейса – наверняка банки пользуются телекоммуникационными услугами, которые предоставляются им независимыми провайдерами.

Функциональность, обеспечивающая взаимодействие с человеком

Обычно информационные системы предусматривают взаимодействие с широким спектром различного типа пользователей. Для примера с банком можно предложить следующий список, в котором перечислена бóльшая часть типов пользователей:

- *Клиенты и пользователи* – должны иметь возможность пользоваться банкоматами и on-line доступом к web-ресурсам банка без какого-либо предварительного обучения; интерфейсы пользователя должны быть интуитивно понятными.
- *Сотрудники банка, работающие с клиентами* – должны иметь возможность оперативно управляться с системой, чтобы обеспечивать быстрое и эффективное обслуживание клиентов из очереди. Эти сотрудники должны быть хорошо обучены и подготовлены. Важная характеристика интерфейса – он должен быть «заточен» под то, чтобы обученные работники банка могли выполнять операции максимально быстро и эффективно.
- *Руководители различных уровней* – некоторые руководители могут не иметь компьютерных навыков, чтобы «справляться» с системой (за исключением случаев, когда руководители назначаются из числа сотрудников, кто ранее работал с системой). Возможности, которые система предоставляет руководству, могут быть теми же, что и для сотрудников обслуживающих клиентов; однако, наряду с этими возможностями руководству необходимо получать и более широкий спектр данных, чтобы видеть общую картину деятельности банка. К таким данным можно отнести консолидированную информацию о работе филиала или суммарные данные по одному из направлений работы банка. Необходимо отметить, что получение такой консолидированной информации требует сбора и хранения информации в течение определенного периода времени, чтобы позволить анализировать тенденции и проводить исторический анализ работы банка.
- *Сотрудники маркетинговой службы и отдела регламента* – для этого типа пользователей могут потребоваться самые различные возможности, например, доступ к полной функциональности системы для начала работ над новым продуктом или проектом.
- *Обслуживающий персонал системы* – должен иметь возможность проводить обновления и модернизацию системы. В идеальном случае должна иметься возможность делать это в любой момент, в том числе, и во время нормального функционирования системы. Но на практике, обновления системы обычно проводят в нерабочее время (ночью), для того, чтобы не мешать работать банку и обеспечить целостность данных.

Защитная функциональность

Защищенность от несанкционированного доступа является суперважной характеристикой банковских систем. Ключевым элементом является сохранение неприкосновенности информации, поскольку она является сердцем бизнеса.

Для обеспечения безопасного доступа к данным и деньгам через банкоматы используются PIN-коды (Personal Identification Number) пластиковых карт. Для обеспечения безопасной передачи данных между банком, его офисами и банкоматами используется шифрование данных.

Другая ситуация, которую необходимо также принимать во внимание, – это обеспечение функционирования системы в аварийных ситуация при сбоях компьютерного оборудования, программного обеспечения, отключения электропитания или обрывах в сетях передачи данных или линиях связи. Эти категории функциональности тесно связаны с оценкой риска. Степень защиты, которую можно себе позволить, чтобы смягчить последствия рисков, серьезно зависит от вероятности их появления.

В заключении – и как самое важное - следует отметить функциональность по защите информационной системы от преднамеренного несанкционированного доступа, проще говоря, от хакеров, расхитителей и других мошенников. Программное обеспечение должно обеспечивать адекватную защиту банка и его клиентов от подобных «неприятностей».

Системные транзакции

Для банковской системы каждый из типов пользователей будет являться заинтересованной стороной. Следовательно, для каждого типа пользователей будет разработан свой набор сценариев использования. Для клиентов банка набор сценариев будет отражать регулярно используемые услуги банка - снятие денег со счета, зачисление денег на счет, переводы средств, сделанные самим клиентом или автоматически в случае, например, зачисления на счет заработной платы, регулярные выплат по кредиту. Очевидно, что будут существовать и не так часто выполняемые транзакции, например, получение ссуды или ипотечного кредита.

Для каждого типа пользователя следует оценить предполагаемую нагрузку, им создаваемую, для того, чтобы оценить время отклика системы на воздействие с его стороны. Разумеется, это не будет фиксированный параметр, поскольку на этот показатель влияет общая загрузка системы, которая, в свою очередь, неравномерна в течение суток, а также в разные дни недели и т.д.

Возрастающая популярность web-сервисов также должна быть принята во внимание при расчете показателей загрузки системы.

Режимы функционирования системы

Доминирующим режимом функционирования системы должен быть нормальный режим. Однако необходимо предусмотреть и другие режимы, которые предусматривали бы следующие возможности - обучение пользователей, резервное копирование данных системы, аварийное восстановление системы и данных, обновление и модернизация системы.

6.2.4 Абстрактная модель автомобиля

Второй пример связан с более «физической» по своей природе системой, но, тем не менее, интересно отметить, что и тут присутствуют те же самые категории информации, хотя и в совершенно другой форме.

Самый сложный вопрос этого примера – будет ли системная модель оставаться физической моделью, и в какой все-таки степени эта модель может быть абстрактной. Маловероятно, что новый автомобиль будет радикально отличаться от своих предшественников – те же четыре колеса будут располагаться по углам корпуса, будут присутствовать двигатель, коробка передач, подвеска, лобовое стекло и т.д.

По этой причине системная модель автомобиля наверняка будет напрямую ссылаться на физические элементы его конструкции, как показано на рис. 6.7. Направление стрелки показывает направление «влияния» одного элемента на другой. Например, водитель нажимает на педаль тормоза, которая задействует тормоза. Стрелки между кузовом автомобиля и другими его частями, которые закреплены на нем, изображены

двунаправленными, чтобы продемонстрировать зависимость в обоих направлениях, т.е. двигатель закреплен на кузове автомобиля, а кузов выполнен таким образом, чтобы на нем можно было закрепить двигатель.

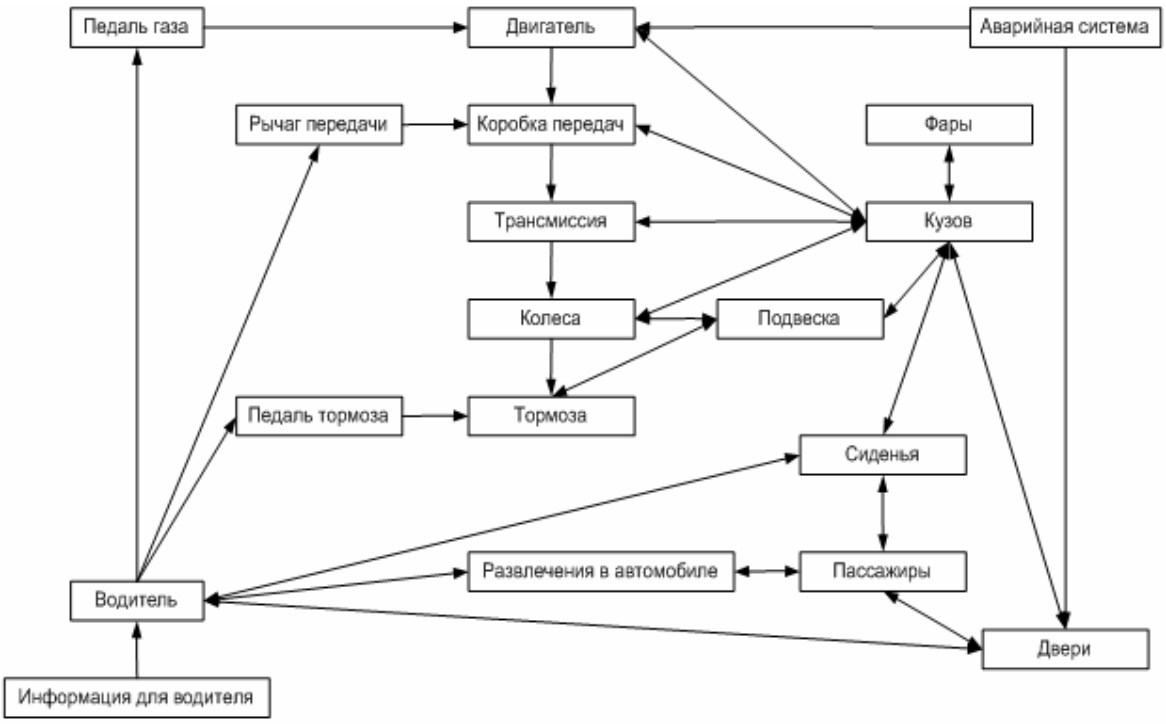


Рис. 6.7 Абстрактная модель автомобиля, базирующаяся на физических элементах.

Однако там, где составляющие нового автомобиля предполагают коренным образом отличаться от традиционного взгляда, например, электронная система управления, которая будет разрабатываться заново, имеется возможность сохранить более абстрактный подход к моделированию, что позволит обнаружить значительные преимущества при поисках наилучшего решения.

Для того чтобы стало ясно, что все вовлеченные в проект специалисты однозначно понимают функциональность автомобиля, необходимо определить качественные характеристики этого термина. Например, понятно, что автомобиль должен перевозить из одного места в другое людей и их багаж, поэтому ключевые вопросы, которые необходимо задавать, чтобы сформировать пользовательского требования, могут выглядеть следующим образом:

- Сколько людей будет перевозить автомобиль?
- Какой объем багажа будет перевозиться?
- Какой уровень комфортабельности должен иметь автомобиль?
- С какой максимальной скоростью может двигаться автомобиль?
- Какой быстро должен разгоняться?
- Сколько должен стоить?
- Какая информация должна отображаться на приборах?

- Какая развлекательная техника должна быть в автомобиле?
- Какими устройствами безопасности должны быть оборудован автомобиль?
- В каких условиях будет эксплуатироваться автомобиль?

Внутренняя функциональность

Ключевые параметры, которыми должен характеризоваться автомобиль на функциональном уровне, включают следующее:

- *Показатель ускорения автомобиля*. Это требует сохранения определенного баланса между мощностью двигателя, суммарным весом автомобиля, коэффициентом обтекаемости кузова, но с учетом параметров колес как ограничений.
- *Класс автомобиля*. Это требует соблюдения баланса между расходом горючего на 100 км, объемом двигателя, типом коробки передач (автоматическая или механическая) и т.д.
- *Уровень комфортабельности автомобиля*. Уровень комфортабельности влияет на стоимость и вес автомобиля; к тому же разные люди могут по разному относиться к оценке различного рода удобств.

Необходимо сразу отметить, что все перечисленные ключевые характеристики не являются независимыми. И это достаточно типичная ситуация для инженерных систем. Наличие взаимосвязей такого рода порождает большое желание «переместить» модель в более абстрактную область.
Например, все перечисленные выше факторы, будут разниться в зависимости от типа двигателя и типа используемого горючего. Для каждого из типов горючего – бензин, дизельное топливо, газ, - такие параметры типа двигателя, как расход топлива, вес двигателя, объем топливного бака, будут иметь разное значение, а значит, будет необходимо решать задачу:

- стоит ли делать окончательный выбор всех параметров сейчас;
- стоит ли оставить этот вопрос открытым на будущее;
- стоит ли предоставить заказчику (покупателю) возможность выбора одного, двух или трех возможных вариантов.

От принятого решения будет в значительной степени зависеть ход дальнейших работ по проектированию автомобиля. Вполне возможно, что придется проектировать, испытывать и оценивать все три возможных варианта, причем даже более детально, нежели общую модель. Хотя возможен и другой вариант, например, выбор газа в качестве топлива будет исключен с самого начала.

Интерфейсная функциональность

Некоторые наверняка подумают, что автомобиль является достаточно независимым устройством с точки зрения его взаимодействия с внешними системами. И совершенно напрасно! Вспомните самый простой пример - в автомобиле есть радиоприемник, который должен удовлетворять определенным стандартам (иметь интерфейс), чтобы взаимодействовать с внешними системами - принимать и декодировать радиосигналы.

Возрастание сложности окружающих нас технологий увеличивает и число стандартов, которым должно удовлетворять создаваемое устройство.
Например, если автомобиль имеет встроенную систему глобального позиционирования (GPS), то появляется необходимость, чтобы этот прибор правильно принимал и декодировал сигналы тех спутников, от которых зависит его работа. Если же на автомобиле установлена система, информирующая водителя о ситуации на дорогах, то эта система должна «уметь» получать информацию от внешних систем, обеспечивающих доставку такой информации. Отсюда вполне можно предположить, что в скором будущем эти системы будут взаимодействовать друг с другом, а, следовательно, будет необходимо поддерживать еще один межсистемный (внутренний) интерфейс.

Для современных автомобилей большую роль играет техническое обслуживание. Очень важно, чтобы автомобиль умел «возвращать» информацию обо всех событиях, которые происходят внутри него во время эксплуатации, чтобы автомеханики могли провести диагностику и обнаружить те части автомобиля, которые функционируют неправильно или срок эксплуатации которых подходит к концу. Это пример системы диагностики, которая частично установлена на функционирующей системе (автомобиле), а частично в сервисном центре, где осуществляется обслуживание автомобиля, и тут важно, чтобы обе составляющие понимали друг друга.

Функциональность, обеспечивающая взаимодействие с человеком

Многие характеристики «пользовательского интерфейса» автомобиля формировались исторически в течение долгих лет. Например, традиционное расположение педалей управления одинаково во всем мире (педаль газа - справа, педаль тормоза - слева, если есть педаль сцепления, то она располагается еще левее).

Другие характеристики, такие, как правое или левое расположение руля, расположение приборов на передней панели или щеток стеклоочистителя (дворников), зависят от местных «традиций» и страны.

С другой стороны, пока не существует общепринятых стандартов и соглашений для систем, предназначенных для развлечения, навигационных устройств и других не очень широко распространенных систем. Следовательно, проектировщики имеют полную свободу при разработке пользовательских интерфейсов для этих систем. Но, как и для большинства электронных систем, пользовательские интерфейсы этих устройств должны быть просты и интуитивно понятны любому человеку, который захочет ими воспользоваться. Это сама по себе достаточно непростая задача, поскольку единственный способ объяснить человеку, как общаться с системой, - это снабдить его инструкцией или руководством пользователя. Совершенно понятно, что невозможно отправить всех водителей или пассажиров на обучающие курсы, так же как совершенно невозможно даже примерно оценить среднестатистический образовательный уровень или опыт человека, желающего воспользоваться системой.

Защитная функциональность

Основной задачей защитной функциональности является обеспечение безопасности автомобиля, водителя и пассажиров. Другой важной задачей является обеспечение защиты автомобиля от грабежа и угона.

В общем и целом, защитная функциональность начинается с тормозной системы. Поэтому очень важно в любой момент времени обеспечить водителю гарантированную возможность воспользоваться эффективной системой торможения. Одним из возможных вариантов обеспечения этого требования является установка на автомобиль тормозной системы с двумя независимыми гидравлическими контурами, которая обеспечивает торможение даже в том случае, когда выходит из строя один из ее элементов.
Системная модель может либо достаточно подробно описывать такую тормозную систему, либо описывать только сам факт наличия у автомобиля тормозной системы. В дальнейших проработках модели даже требование того, что тормозная система должна оставаться работоспособной в случае выхода из строя одного из гидравлических контуров, следует рассматривать вне рамок модели.

Попутно заметим, что это обсуждение автоматически подразумевает тот факт, что используется именно гидравлическая тормозная система! Таким образом, некоторые технические решения могут появляться на самых ранних этапах, если существует хорошо наработанная практика или если решение принимается исходя из поставленных бизнес-целей, которые заранее определены во входящих требованиях.

Другие аспекты обеспечения защиты включают антиблокировочную систему торможения ABS и подушки безопасности, смягчающие удар при столкновении. Опять же, данные функции могут быть либо сразу внесены в системную модель, либо этот вопрос может быть отдан на откуп дизайнеру с тем, чтобы создать нечто принципиально новое.

Для обеспечения защиты автомобиля от грабежа и угона необходимо, как минимум, предусмотреть замки на дверях. В дополнение к этому автомобиль может быть оборудован противоугонной сигнализацией и иммобилайзером двигателя. Ограничивающим фактором для такой дополнительной функциональности является лишь ее стоимость (сколько покупатель готов платить за нее). Однако, существуют и стимулирующие факторы, например, – наличие аналогичной функциональности в автомобилях-конкурентах, положительное отношение к этой функциональности со стороны страховых компаний и т.п. Все эти факторы оказывают исключительное влияние не только на то, будет ли присутствовать данная функциональность в автомобиле, но и насколько оправдано наличие такой функциональности.

Системные транзакции

Существует достаточно большое количество возможных транзакций в отношении автомобиля. В основе всех их лежат поездки и путешествия, но каждая со специфическими целями и характерными особенностями, например:

- Водитель, поездка за покупками в город – покинуть парковку, поехать, припарковаться, поставить на сигнализацию, открыть автомобиль, загрузить автомобиль, покинуть парковку, поехать, припарковаться, разгрузить автомобиль, поставить на сигнализацию.
- Водитель, поездка по автостраде.
- Водитель, поездка в аэропорт (с багажом).
- Водитель, поездка с аварией.
- Пассажир – сесть в машину, пристегнуть ремень безопасности, ехать, расстегнуть ремень безопасности, выйти из машины.
- Автомеханик – проводить периодическое обслуживание с большими/малыми интервалами.

- Владелец – купить, период старения/удешевления автомобиля, продать/подарить.
- Продавец – периодически пытаться продать, продать, гарантийный период.

Каждая из этих транзакций может формировать новые требования, такие как объем багажника или средства для технического обслуживания. Следовательно, важно рассмотреть все транзакции и понять, какие требования вытекают из каждой. Конечно, это вовсе не означает, что все эти требования будут удовлетворены. Возможно, что некоторые из них будут отклонены по причине высокой стоимости их реализации или потому, что они не востребованы рынком, для которого предназначен автомобиль. Как возможная альтернатива может быть принят промежуточный вариант, при котором для различных географических рынков будут создаваться различные модели автомобиля.

Режимы функционирования системы

Нетрудно себе представить, что характеристики местности, в которой предполагается эксплуатировать автомобиль, могут влиять на его конструкцию. Например, в горной местности коробка передач могла бы автоматически выбирать пониженные передачи, а система управления двигателем могла бы учитывать пониженное содержание кислорода в атмосфере для подготовки смеси, впрыскиваемой в цилиндры двигателя.
Такие возможности можно или с самого начала предусмотреть в базовом варианте поставки и тогда водитель мог бы использовать этот режим вождения по мере необходимости, или же, совершая покупку, водитель должен будет отдельно оговаривать наличие этой опции.
Другой важный режим функционирования - это возможность диагностики. Система контроля за работой двигателя должна «уметь» передать накопленные данные системе диагностики сервисного центра для последующего анализа.

Более экстремальным режимом функционирования является автопоезд. В этом случае вся сцепка будет управляться с головного автомобиля, а средства управления во всех остальных будут отключены.

6.2.5 Получение требований из системной модели

Создание структуры документа для требований

Как отмечалось ранее, системная модель может состоять из нескольких независимых и потенциально перекрывающихся моделей. Можно начинать формирование требований из любой из разработанных моделей, как это пояснялось в предыдущих разделах на примерах модели банка и автомобиля. Однако проблема состоит в том, что достаточно сложно сразу получить хорошую структуру для требований, в которой каждое требование однозначно занимало бы свое исконное место. И при этом каждое свободное место в этой структуре означало бы некое поле для маневра дизайнерской мысли, а не присутствовало потому, что что-то упущено. Разработка структуры документа уже рассматривалась нами в главе 4.

Рекомендуется выбрать в качестве базовой одну из моделей для последующей разработки структуры документа. Выбранная модель должна быть максимально общей, поскольку системные требования должны описывать всю систему целиком, а не только какую-нибудь ее маленькую часть. На практике такой выбор достаточно очевиден и не вызывает больших проблем. Для систем обработки информации, как приведенная

банковская система, модель данных зачастую является наилучшим выбором, потому что покрывает большую часть общей функциональности системы, в той или иной степени, отражая сбор данных, их распространение, обновление и обеспечение безопасности. Для систем более «физических» по своей природе, например, для примера с автомобилем, обычно лучше всего использовать модель, отражающую физическую структуру системы (если она существует), поскольку большинство требований будут относиться к тому или иному физическому элементу системы.

Получение и распределение требований

Как только разработана и согласована структура требований, начинается процесс сбора производных требований и размещения их в полученной структуре. Вполне возможно, что некоторые входящие требования могут быть сразу и без изменений помещены в структуру документа. Если это происходит, то обычно свидетельствует о том, что входящие требования весьма детализированы, т.е. очень близки к реализации.

Если же вы только начинаете формулировать требования, то самое лучшее – это руководствоваться всеми правилами для «написания хороших требований», приведенными в главе 4. Всегда помните и соблюдайте «золотое правило» - каждое требование должно представлять собой единственное утверждение, реализацию которого можно проверить (т.е. каждое единичное требование должно быть пригодно для тестирования). Как только производное требование сформулировано, необходимо установить связь, идущую от него обратно к одному или более входящим требованиям, которые данное (только что сформулированное) требование удовлетворяет полностью или частично.

Когда мы рассматриваем пригодность требования для тестирования, имеет смысл сразу задуматься о критериях, которые будут характеризовать пройден тест успешно или нет. Эти критерии приемки должны быть зафиксированы для каждого требования. В некоторых случаях такой критерий может включаться непосредственно в текст требования, как, например, характеристика производительности. Но в основном рекомендуется отражать приемочный критерий в отдельном атрибуте конкретного требования.

После определения критериев приемки (и производительности) необходимо рассмотреть способы, с помощью которых будет демонстрироваться тот факт, что система удовлетворяет этим приемочным критериям. Иными словами, необходимо определить критерии приемки, стратегию проверки, разработать необходимые программы испытаний и инспекций.

На том этапе необходимо также оценить степень трудоемкости тестирования и определить, какое тестовое оборудование потребуется для испытаний. Может вполне оказаться, что организация тестирования потребует отдельных разработок, а в некоторых случаях, даже запуска отдельных (дополнительных) проектов.

Дальнейшее пристальное внимание к этому вопросу может привести вас к идее встроить в систему тестовую функциональность и организовать контрольные точки для мониторинга. Встроенная тестовая функциональность имеет очень большое значение особенно в тех областях, которые связаны с обеспечением безопасности. Например, в случае с автомобилем, - сразу после запуска двигателя электронные системы автомобиля тут же выполняют диагностику двигателя и всех вспомогательных подсистем. Контрольные точки встраиваются в систему так, чтобы ранее недоступная важная информация о системе, стала доступной и возможной для обработки. Простой пример – наличие в автомобиле прибора для измерения давления топлива. Если обратить внимание на информационные системы, -

вспомним пример банка – то можно говорить о специальном дисплее, на котором отображалась бы консолидирующая информация о количестве транзакций в единицу времени для всей сети банка.

Заключительным набором требований, который также следует рассмотреть, являются ограничения, которые не добавляют дополнительной функциональности, но ограничивают способ, которым может быть реализована необходимая функциональность. На уровне системных требований часть ограничений может без изменений переноситься из пользовательских требований. Например, геометрические параметры системы могут быть жестко заданы заинтересованными сторонами, или заказчики настаивают на том, что одна из существующих систем должна входить в виде подсистемы в новую систему.

Могут существовать также и некоторые другие источники ограничений:

- ***Архитектурные решения*** – например, решение о том, что на автомобиле должна быть установлена гидравлическая тормозная система с двумя независимыми контурами.
- ***Область применения*** – например, все оборудование, установленное на автомобиле, должно работать в условиях вибрации, которая присутствует при движении автомобиля.
- ***Безопасность*** – например, каким образом разработчики могут убедить компетентные органы в том, что данный автомобиль не представляет угрозы для других участников движения?
- ***Производство*** – например, может ли создаваемый автомобиль производиться на имеющихся производственных мощностях и с приемлемыми затратами?

6.2.6 Согласование системных требований с проектировщиками

Заключительным шагом разработки системных требований является согласование их с командой проектировщиков, которые будут отвечать за дальнейшее проектирование.
Так как эта тема уже рассматривалась в главе 2, то нет никакой необходимости в дополнительных подробных объяснениях.

6.3 Получение требований для подсистем из системных требований

Следующим логическим шагом, который необходимо выполнить после разработки системных требований, является разработка архитектуры системы, чьими компонентами являются крупные подсистемы (см. рис. 6.8).
Как обычно, данный процесс начинается с согласования с заказчиком входящих требований. В качестве основы для процесса согласования системных требований должны применяться те же критерии оценки, которые уже были описаны в главах 2 и 4. Требования не должны «скатываться» в область детальных технических решений, если только нет специфической необходимости каким-то образом ограничить дизайн. Да и в этом случае, такие требования должны быть явным образом сформулированы как ограничения. Зачастую единственным оправданием существования ограничения является фраза - «потому что на этом настаивал заказчик». Поэтому хорошей практикой является пересмотр ограничений («подвергать сомнению»), особенно если ограничение в большей степени

подразумевается, нежели напрямую и явно сформулировано. Порой требования для подсистем могут быть выражены в терминах определенных технических решений, что, в какой-то мере, демонстрирует лень разработчиков, которые склонны к преждевременной детализации.

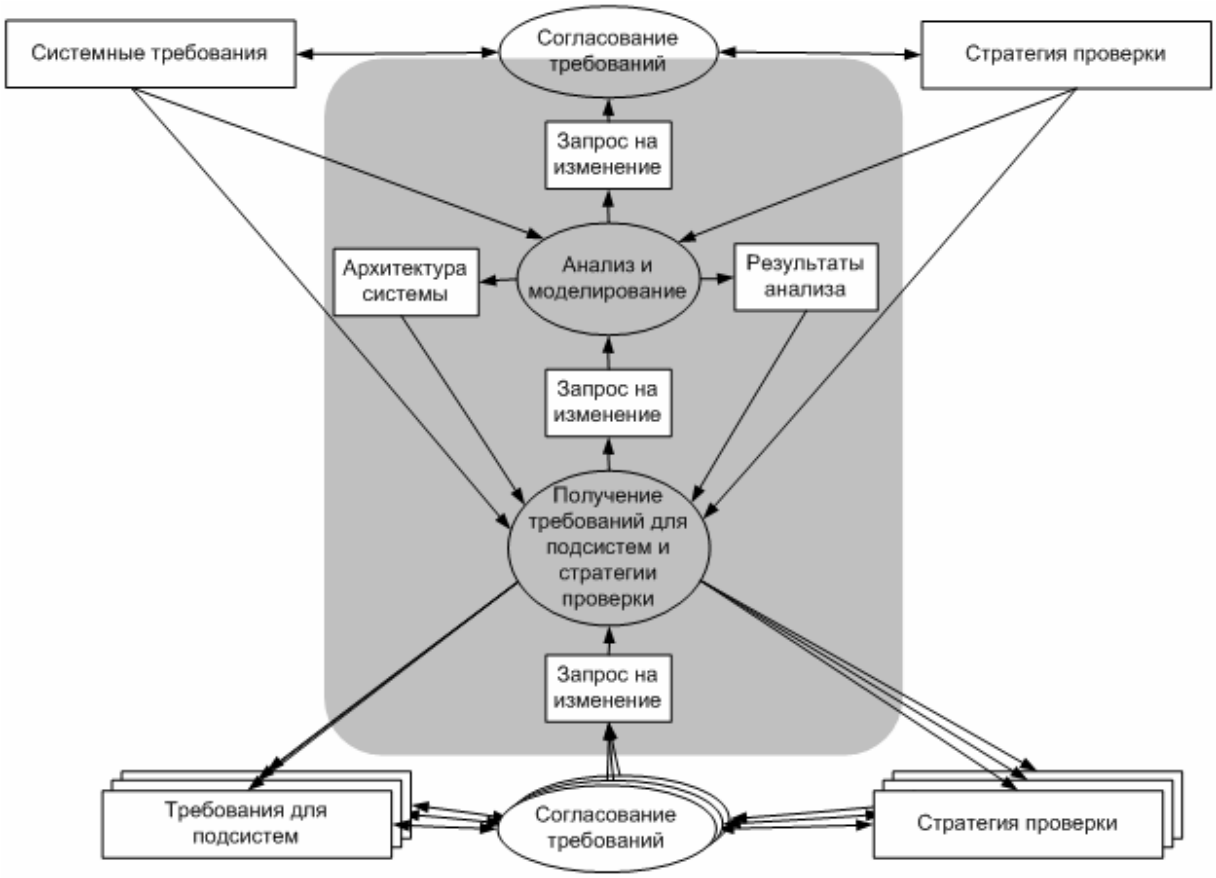


Рис. 6.8 Основной процесс разработки требований для получения требований для подсистем из системных требований.

Аналитическая работа, необходимая для поддержания процесса согласования требований, помогает проектировщикам лучше понять, для чего предназначена система и подсистемы, и начинать задумываться о возможных реализациях.

6.3.1 Разработка архитектурной модели системы

Архитектурная модель системы идентифицирует компоненты системы и то, как они между собой взаимодействуют. Проектировщик системы должен хорошо понимать, как взаимодействие компонентов (подсистем) приводит к появлению требуемых системных свойств, т.е. каким образом они удовлетворяют входящим требованиям.
Проектировщики должны быть способны прогнозировать появление свойств, которые, может быть, и не требовались по определению, например, возможные вредные воздействия,

влияющие на безопасность и окружающую среду. Однако вполне возможно появление и таких «побочных» системных свойств, которые - хоть и не требовались, - могли бы быть приемлемы для дальнейшего использования. Такие свойства следует обсудить с заказчиком. На усмотрение заказчика эти дополнительные возможности могут быть либо прописаны во входящих требованиях (т.е. «задним числом» востребована их реализация), либо, наоборот, заказчик может настаивать на их исключении (запрет на проявление таких свойств).

Вполне возможен и такой случай, когда проектировщики выяснят, что не существует способа удовлетворить требования вообще или же в рамках определенного бюджета проекта.

После того, как разработана и проанализирована архитектура системы, появляется свет в конце туннеля. Наличие такого дизайна дает теперь возможность с намного большей степенью точности и аккуратности оценить затраты и время, которые потребуются на реализацию системы. А это, в свою очередь, дает возможность провести раунд переговоров с заказчиком с целью уточнения тех входящих требований, реализация которых в отведенное время или за отпущенные деньги может вести к проблемам.

Во многих случаях имеет смысл разработать несколько вариантов реализации с тем, чтобы затем проанализировать и оценить преимущества каждого из них. Это также может привести к очередному этапу переговоров с заказчиком для определения наиболее подходящего варианта реализации с точки зрения достижения лучшего соотношения «получаемые преимущества \ затраты на реализацию».

После согласования архитектуры каждый компонент должен быть описан в терминах его внутренней функциональности и его «обязанностей» взаимодействовать с другими компонентами и внешними системами.

6.3.2 Получение требований из архитектурной модели системы

Требования для компонентов (подсистем) могут быть получены из их описания, которое было разработано на этапе моделирования системной архитектуры. Требования должны описывать функциональность, обеспечиваемую компонентом, интерфейсы, которые он должен использовать или поддерживать, а также ограничения, налагаемые на него. Ограничения на компонент могут *накладываться* общими системными ограничениями (например, для всех электронных компонентов системы должна использоваться одна и та же определенная технология), или же ограничения для компонента могут «*вытекать*» из системных ограничений (например, допустимый общий вес системы должен быть строго распределен между ее компонентами). Требования, которые предъявляются к компоненту (или подсистеме), являются, по сути, системными требованиями компонента, если рассматривать этот компонент, как независимую систему со своим собственными правами.

Для каждого производного требования (компонента) необходимо установить связь, ведущую обратно к входящему требованию (системному), которое оно (производное требование) частично или полностью удовлетворяет.

Для каждого компонента необходимо также разработать свою стратегию проверки. Так же, как и для других уровней, здесь важно, чтобы требование было пригодно для проверки. Контролепригодность это один из важнейших аспектов разработки требований, и поэтому стратегия проверки должна разрабатываться по мере окончания разработки дизайна.

6.4 Другие преобразования с использованием архитектуры системы

Поскольку процесс разработки последовательно «спускается» все ниже и ниже от одного уровня к другому, то на каждом из этих уровней (этапов) создается, по сути, собственная архитектурная модель (см. рис. 6.1). На каждом этапе процесс проходит точно так же, как это описано в предыдущих разделах. Другими словами, после разработки требований для подсистем более высокого уровня, начинают разрабатываться требования для компонентов этих подсистем, которые, на следующем шаге, рассматриваются уже как независимые подсистемы со своими компонентами, и так далее.

Однако существуют особый случай (использования архитектурной модели системы), который может считаться исключением из этого правила. Это отражено на рис. 6.9, который показывает, что архитектура системы и последующие требования для подсистем могут быть получены непосредственно из пользовательских требований.

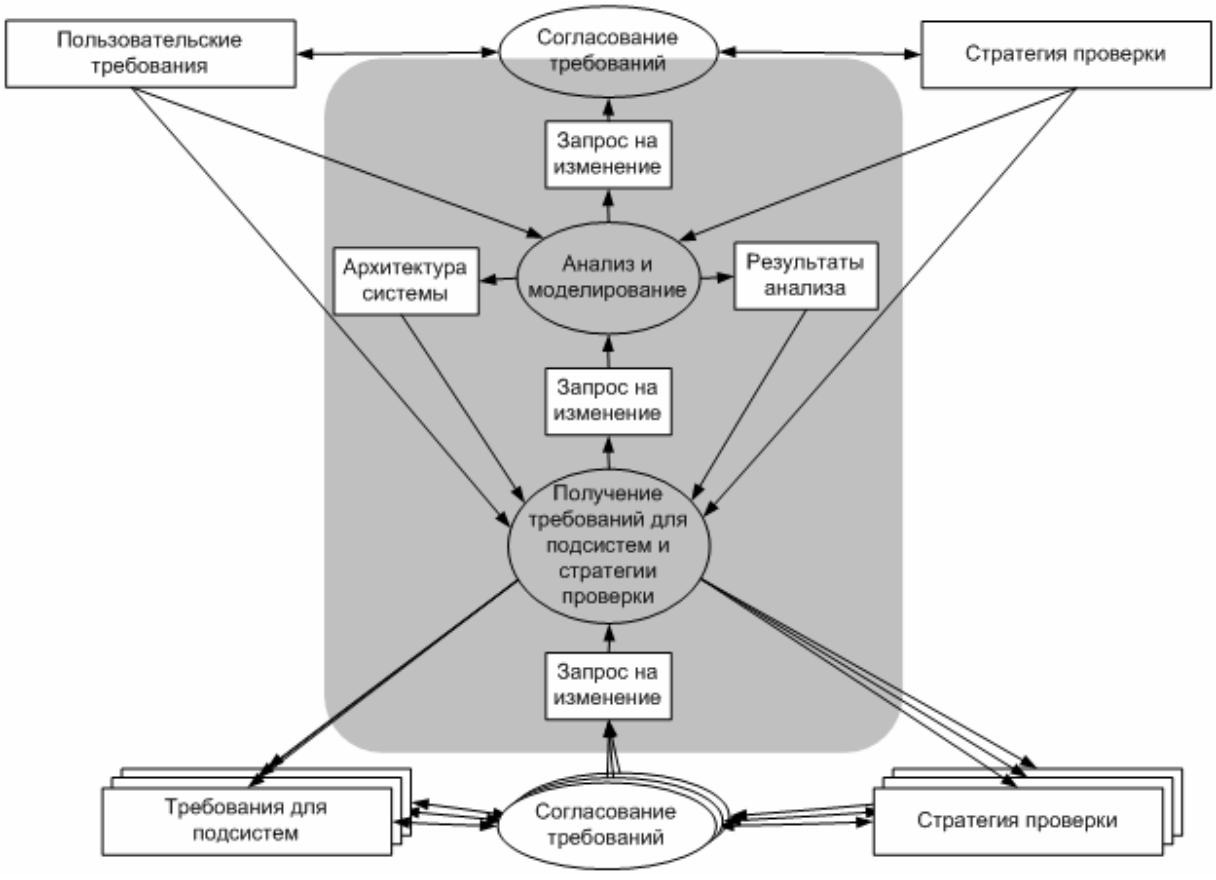


Рис. 6.9 Преобразование пользовательских требований в требования для подсистем.

Такой вариант возможен в том случае, когда архитектурная модель системы известна заранее (вспомните, например, уже ранее рассмотренные «физические» системы - автомобиль или самолет).

Здесь уместно также привести пример из телекоммуникационной индустрии, где применяемые стандарты зачастую предписывают архитектуру решения. Опять же встает чисто теоретический вопрос, являются ли требования, предъявляемые к системе, на самом деле пользовательскими или они, по сути своей, являются уже системными требованиями. Какой бы ответ ни был бы получен, это вполне обычная практика – транслировать входящие требования на уровень производных (требований к подсистемам). Получается это благодаря тому, что стандарты, сами по себе, достаточно детализированы.

6.5 Заключение

В данной главе были рассмотрены общие свойства области решений и охарактеризован процесс разработки требований, используемый в этой области.

Было дано описание процесса преобразования пользовательских требований в системные и последующий процесс преобразования системных требований в требования для подсистем.

Были рассмотрены два различных примера, которые помогли раскрыть типы функциональностей, используемых для описания требований в области решений.

Было продемонстрировано, что помимо внутренней функциональности необходимо рассматривать также и интерфейсную функциональность; функциональность для обеспечения взаимодействия с людьми; а также защитную функциональность, обеспечивающую как защиту системы от внешних воздействий, так и безопасность ее эксплуатации по отношению к людям и окружающей среде.

Было упомянуто, что на более поздних этапах возможно появление дополнительных ограничений, в том числе, для соблюдения требований применяемого законодательства и для прохождения системой необходимой сертификации.

7 Расширенные связи и их анализ

Вначале есть «Кто?», «Что?», «Когда?» и «Где?»,
затем последуют «Куда?», «Откуда?», «По какой причине?»
и очень-очень много «Почему?».

Зафод Библброкс из книги «Автостопом по Галактике».
Дуглас Ноэль Адамс (Douglas Noel Adams),
писатель, 1952–2001

7.1 Введение

Очень часто основная суть конкретного архитектурного решения или глубокое понимание того, каким образом компоненты системы взаимодействуют друг с другом для достижения поставленной задачи, остается в головах инженеров. Спустя несколько месяцев или лет, когда аналитики и проектировщики системы уже «ушли» с проекта или заняты чем-то другим, когда из их памяти стерлись многие нюансы, потеря этой информации может создать серьезную проблему для последующей модернизации системы, ее обслуживания или ее повторного использования (частичного или полного).

Вот почему в первую очередь мы рассмотрим в данной главе методологию, позволяющую сохранить эту информацию путем фиксирования ее с помощью связей между проблемой, спецификацией решения и архитектурой. Этот метод, имеющий название «расширенный анализ связей», базируется на концепции «элементарных связей», которая описывалась нами в главе 1 и использовалась на протяжении всех последующих глав книги.

Содержание этой главы без труда подскажет ответы на «очень-очень много “Почему?”», а вот для того, чтобы ответить на вопросы «Куда?», «Откуда?», «По какой причине?», которые относятся к характеристикам связей, мы рассмотрим в этой главе другую тему – метрики и параметры связей.

7.2 Элементарные связи

Существует много способов отображения связей типа «многие-ко-многим». Вот вам случай из реальной жизни.

Однажды к подрядчику оборонного ведомства пришел консультант для проверки наличия и качества связей в проектной документации. Он застал следующую ужаснувшую его картину... На полу огромного кабинета во всю его длину лежали две полосы бумаги, на одной из которой были распечатаны все требования, а на другой полосе, – располагавшейся параллельно первой и на некотором расстоянии от нее, - был распечатан исходный программный код. Другие полоски бумаги, на которых было что-то напечатано и которые располагались так, чтобы соединить обе параллельные ленты бумаги, обозначали связи между требованиями и отдельными позициями исходного кода.

Эта конструкция занимала невероятно много места, требовала много времени, чтобы ее организовать, это было практически невозможно перемещать, это было невозможно поддерживать в актуальном состояние сколь-нибудь долгое время, но это работало!!!

Многие аналитики и инженеры, возможно, видели связи, представленные в виде таблиц, которые могли оформляться как отдельное приложение к соответствующему документу. Например, для того, чтобы установить связи между пользовательскими требованиями и системными, создают таблицу, в заголовках строк которой записывают номера пользовательских требований, а в заголовках столбцов – номера системных требований. Наличие связи между требованиями обозначается «крестиком» в соответствующих ячейках на пересечении.

Однако у данного подхода есть также ряд недостатков:

- Если количество требований по обеим осям достаточно велико, то лист бумаги или экран компьютера будут слишком малы, чтобы отобразить необходимое количество информации.
- Если связей в таблице не так уж и много, то большинство ячеек окажутся пустыми, что будет, по сути, расточительством полезного пространства.
- Очень тяжело проследить связи через несколько уровней, потому что приходиться работать с несколькими матрицами связей, т.е. фактически переходить от двумерной матрицы к трехмерной.
- Информация о связях требования отделена от сути самого требования.

В качестве альтернативы табличному методу, некоторые используют документы, которые «связаны» гиперссылками (*hyperlinks*). В этом случае требования могут быть выделены, взаимосвязаны с другими требованиями, и можно «гулять» по этим связям в любом направлении, при условии, что вы достаточно сообразительны.
В данном случае информация о связях (traceability) становится уже видимой в тексте требования, но все же и в этом подходе есть свои недостатки:

- Для того чтобы провести анализ связей, вам возможно придется «пройти» весь путь, прежде чем вы увидите текст на другом конце этой цепочки.
- Если на другом конце гиперссылки требование было удалено, то на это конце ссылки у вас нет достоверной информации об этом. Постоянная необходимость отслеживать эти «зависшие» связи весьма затрудняет анализ.

Если вы не используете никакие инструментальные (программные) средства для управления связями, то какой бы метод вы не использовали, координировать эту работу вам будет весьма тяжело.

Простейшую форму управления связями между требованиями могут обеспечить возможности базы данных. В этом случае рекомендуется хранить отдельно сами требования и информацию о связях между ними. Более того, очень важно, чтобы каждое требование обладало собственным уникальным идентификатором, по которому его можно было бы однозначно определить и использовать для установления\поиска связей.

В общем и целом, для правильной организации связей между требованиями (в целях дальнейшего их анализа) необходимо иметь следующие возможности:

- возможность создания связей между требованиями, тем самым формируя «разрешенные» отношения между ними;
- возможность контролируемого удаления связей;
- возможность одновременно видеть текст требований (+ атрибуты) на обоих концах выбранной связи;
- возможность выполнять анализ покрытия (coverage analysis) для выделения тех требования, которые охвачены (*или не охвачены*) данной связью;
- возможность выполнять одно- и многоступенчатый (многоуровневый) анализ влияния (impact analysis) для выделения таких наборов требований, которые оказывают влияние друг на друга;
- возможность проведения одно- и многоуровневого анализа происхождения требования для отображения всей цепочки (дерева) требований: от исходного - до данного;
- возможность проведения восходящего и нисходящего анализа покрытия для выявления требований покрываемых (*или не покрываемых*) выбранной связью.

На рис. 7.1 показан пример элементарных связей. Пользовательское требование связано входящими связями с тремя соответствующими системными требованиями. Можно одновременно видеть и текст пользовательского требования, и текст системных требований, которые его удовлетворяют. Рис. 7.2 демонстрирует второй пример простейших связей. Такой подход существенным образом облегчает анализ связей.

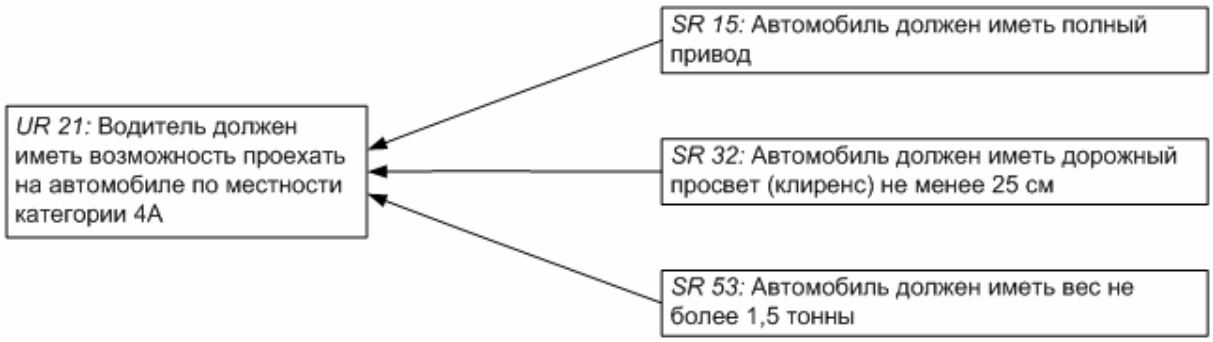


Рис. 7.1 Пример элементарных связей: армейский автомобиль.

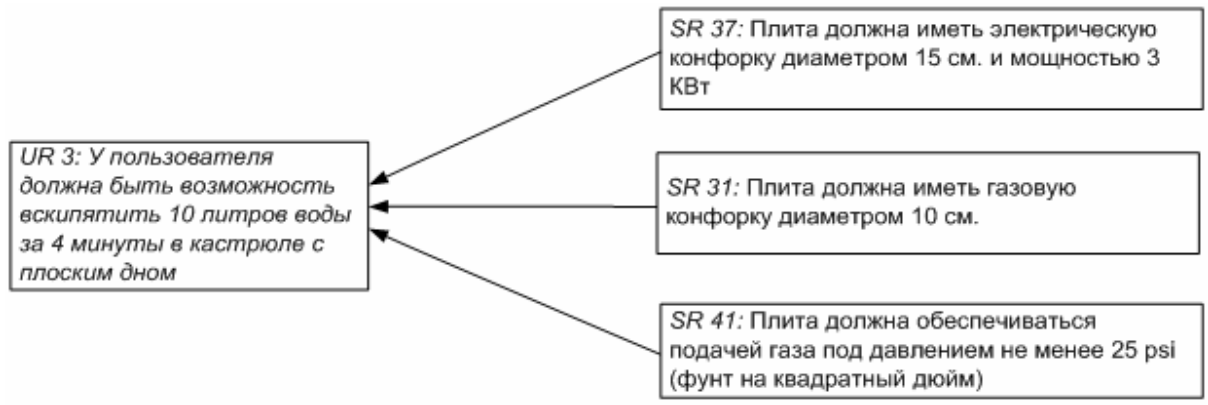


Рис. 7.2 Пример элементарных связей: плита.

7.3 Аргументы удовлетворения

Внедрение практики использования хотя бы элементарных связей, что было рассмотренно в разделе 7.2, для большинства организаций является огромным шагом вперед. И это не ирония, - изменение культуры работы с требованиями внутри организации для внедрения даже такого простого метода, является достаточно большим прорывом. Но, как известно, - нет предела совершенству.

Смысл рис. 7.1 и 7.2 заключается в том, чтобы показать, что три системных требования существуют для того, чтобы удовлетворить одно пользовательское требование. Однако для человека, который не является экспертом в этих областях, достаточно сложно оценить обоснованность этих утверждений. А все потому, что на рисунках никак не отражена аргументация (первопричина), обосновывающая такие решения.

Картина была бы намного лучше, если для каждого требования было бы представлено такое понятие, как «аргумент удовлетворения». Из примера элементарных связей, показанного на рис. 7.1, видно, что три системных требования играют определенную роль в удовлетворении пользовательского требования, а вернее в удовлетоворении какого-то параметра (причины), связанного с этим требованием, но нет ничего, что точно показывало бы, что же это за причина и почему вдруг возникла необходимость иметь именно такую формулировку требования.

Применение расширенных связей позволяет фиксировать такую информацию. В этом случае появляется вспомогательное утверждение (причина), которая на схеме располагается между пользовательским требованием и соответствующими системными требованиями, как это показано на рис. 7.3, и которая носит название «аргумент удовлетворения».

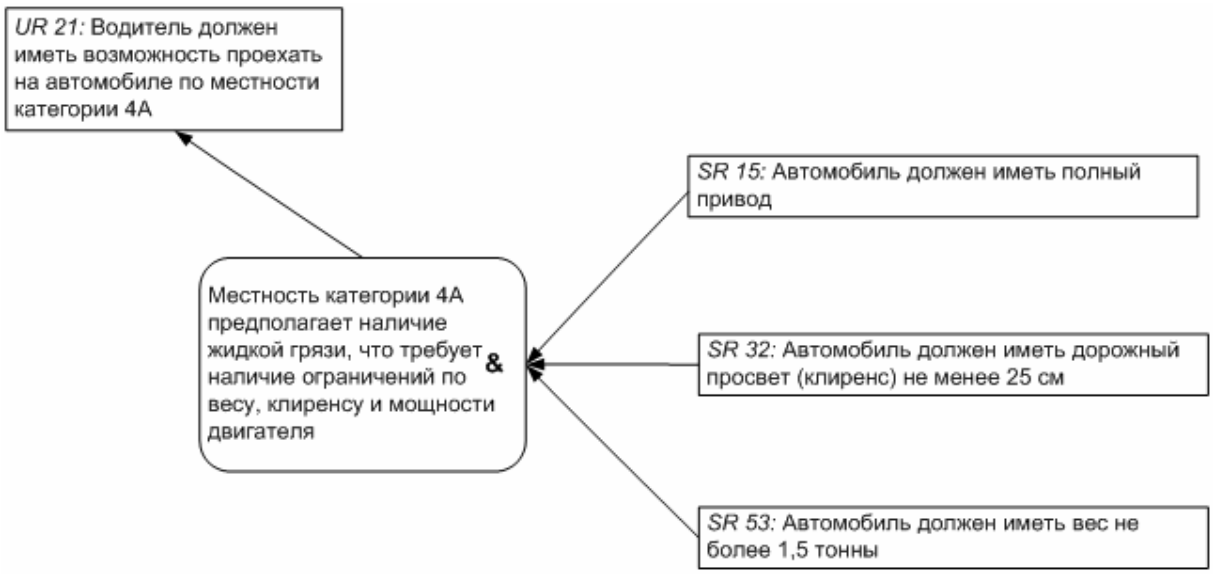


Рис. 7.3 Пример расширенных связей: армейский автомобиль.

Если внимательно взглянуть на рисунок, то нетрудно заметить, что данный подход предусматривает не только текстовое описание аргумента удовлетворения пользовательского требования, но также и отображение того, каким образом для

удовлетворения пользовательского комбинируются системные требования. Для этих целей используется пропозициональный оператор:

- использование конъюнкции (**&** = **и**) (*conjunction*) показывает, что для реализации параметра удовлетворения необходимо выполнение ***всех*** системных требований;
- использование дизъюнкции (**or** = **или**) (*disjunction*) показывает, что для реализации параметра удовлетворения необходимо выполнение ***любого*** из системных требований.

Из примера конъюнкции на рис. 7.3 становится ясно, что пользовательское требование UR 21 может быть удовлетворено только в том случае, если будут реализованы все и каждое из системных требований – SR 15, SR 32 и SR 53.

На рис. 7.4 приведен пример использования дизъюнкции. Пользовательское требование удовлетворяется **или** наличием электрической конфорки, **или** газовой конфорки, **или** и того и другого одновременно. Обратите внимание на применение двухуровневой пропозициональной структуры параметра удовлетворения.

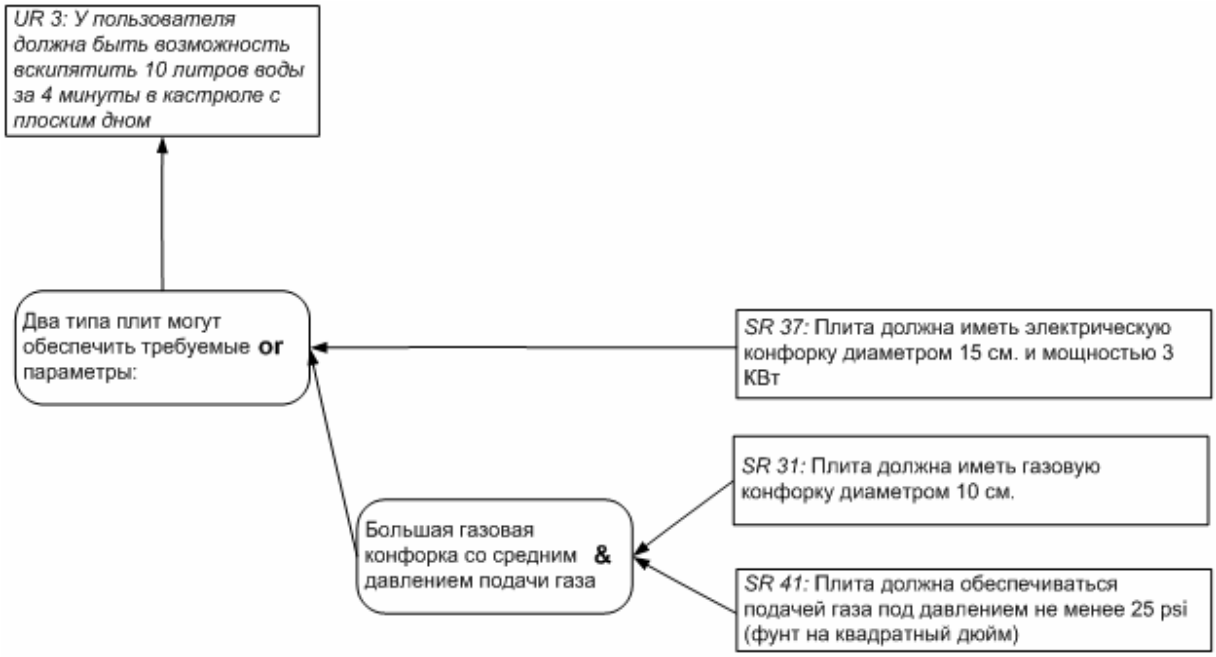


Рис. 7.4 Пример расширенных связей: плита.

Как видите, при таком подходе отображается уже намного больше информации о том, каким именно образом удовлетворяется пользовательское требование. Теперь даже человек, не обладающий глубокими познаниями в предметной области, будет способен оценить важность приведенной аргументации. Текст аргумента помогает оценить логическое обоснование и полноту удовлетворения требования, а операторы делают структуру аргумента более точной.

Теперь вы можете сравнить (и отметить разницу) пример на рис. 7.2 и его развитие на рис. 7.4. Если в первом случае было вовсе и неясно, что системные требования представляют собой два альтернативных решения, то уже на рис. 7.4 это абсолютно

очевидно. Напрашивается и соответствующее решение - в случае, если невозможно будет сделать плиту с электрической конфоркой, ее можно будет заменить на газовую.

С применением концепции расширенных связей авторы впервые столкнулись на проекте модернизации сети железнодорожных дорог западного побережья Великобритании (Network Rail, - в последствии Railtrack - West Coast Route Modernization). Работая над этим проектом, специалисты компании Praxis Critical Systems разрабатывали процесс управления требованиями и моделью данных, используя принцип «обоснования спецификаций» (design justifications).

Этот же самый принцип лежит в основе многих других методов и подходов, в которых аргументы удовлетворения могут иметь разное название, например: «уточнение требований», «обоснование связей», «стратегия» и т.д.

Следует заметить, что аргумент удовлетворения может быть связан не только с требованиями (более низкого уровня). На рис. 7.5 показано, как аргумент удовлетворения дополняется знаниями о предметной области (DK = Domain Knowledge).

В качестве знания о предметной области может выступать либо конкретный факт, либо некоторое предположение о реальном мире, которые не являются, по своей природе, ограничением для возможных решений. В данном случае знание о предметной области является важной частью аргумента удовлетворения (на рисунке показано в виде параллелограмма).

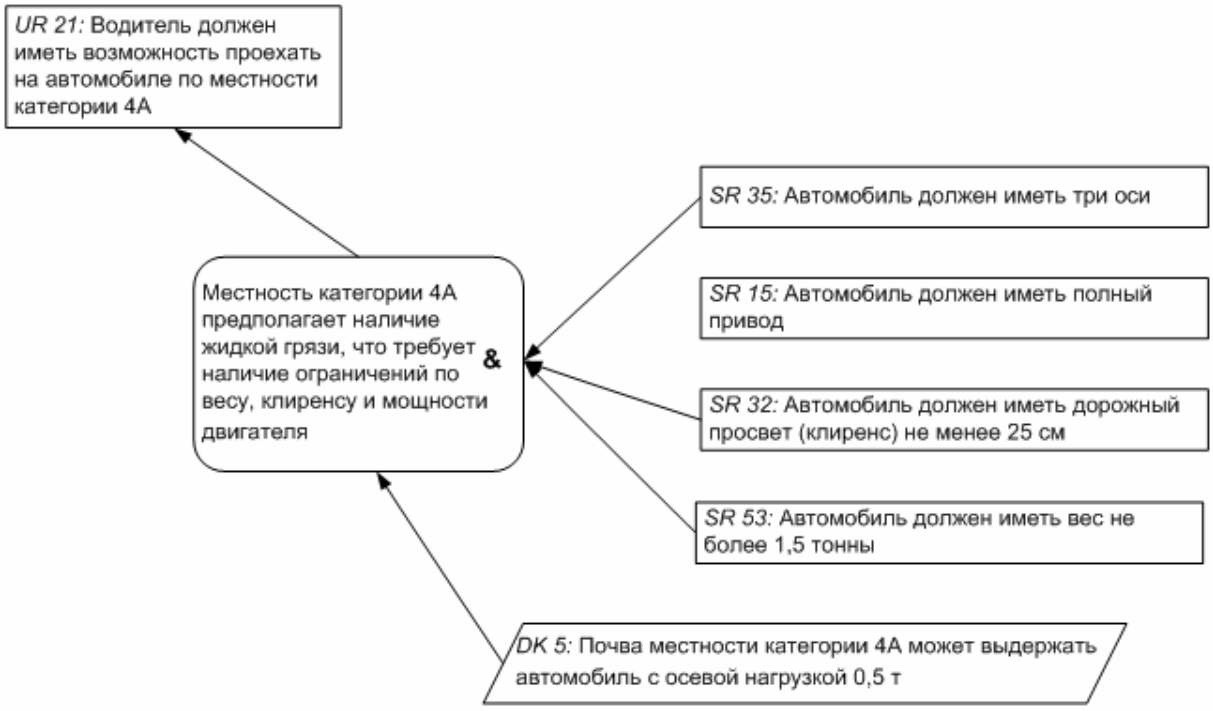


Рис. 7.5 Роль знаний о предметной области.

Сбор и фиксирование подобных предположений и знаний о предметной области является очень важным аспектом разработки требований. Однако окружающая действительность, а значит и наши знания о ней, имеют свойство меняться. Поэтому, зафиксировав определенное предположение, необходимо провести анализ последствий для

того, чтобы оценить насколько система будет удовлетворять заданным требованиям в случае изменения этого предположения.

В качестве примера можно вспомнить серию аварий в Нью-Йоркском метро, которые произошли в 70х годах из-за неверного предположения относительно длины тормозного пути поезда. Изначально, предположение было справедливым, но постепенно поезда становились тяжелее и, как следствие, их тормозной путь увеличивался. А поскольку система сигнализации, хоть и была исправна, но долгое время не модернизировалась, то с некоторого момента времени она перестала удовлетворять изменившимся требованиям.

Возможность документировать и отслеживать степень влияния на систему такого рода предположений становится возможным только при наличии эффективных расширенных связей.

Другой пример, когда не являющаяся частью требований информация играет свою важную роль в удовлетворении аргументов, связан с моделированием. Очень часто аргументы удовлетворения получаются в процессе разработки больших и сложных моделей, но очень часто они детализированы и их не так-то просто уместить в рамки расширенных связей.

Рис. 7.6 демонстрирует часть проекта, связанного с железной дорогой, в котором аргумент удовлетворения зависит от результатов сложного моделирования ж-д расписания, проводимого с помощью специального программного обеспечения.

Весь набор предположений и требований для подсистем был получен в результате моделирования и затем зафиксирован в структуре расширенных связей. Ссылка на результаты моделирования изображена с помощью прямоугольника с закругленными углами.

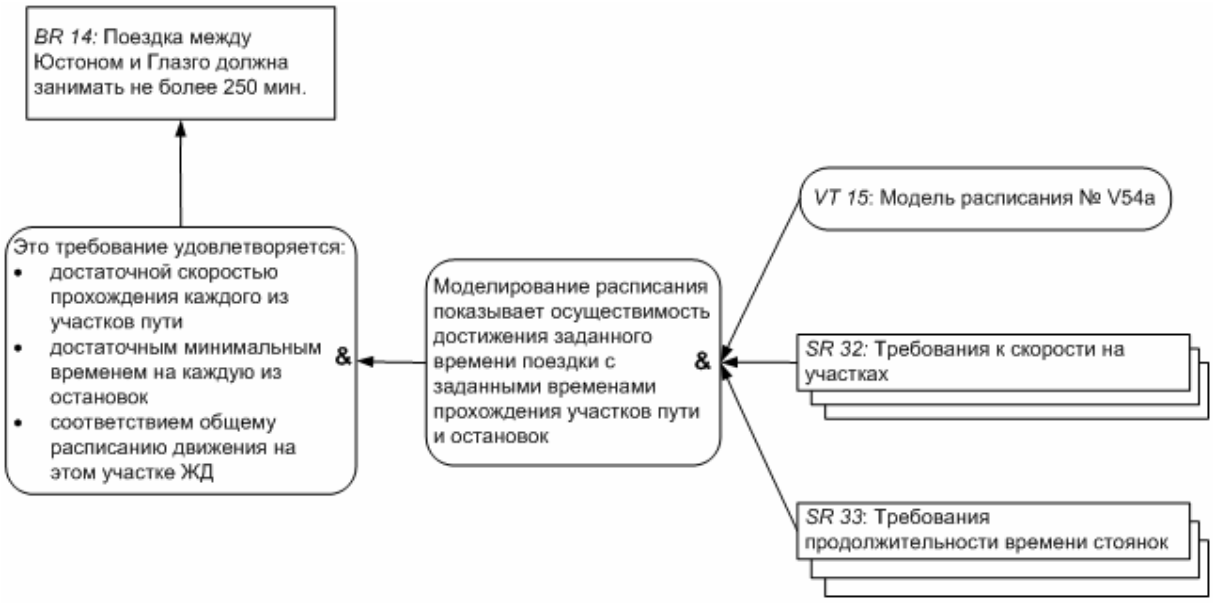


Рис. 7.6 Роль моделирования.

Необходимо отметить, что данный подход имеет еще одно преимущество, которое заключается в том, что с помощью анализа влияния (impact analysis) становится возможным

отследить, как отработка модели и вносимые в нее изменения, влияют на формулировки и структуру требований.

Несомненно, что метод расширенных связей может использоваться не только на одном уровне. Так, на рис. 7.7 изображен пример организации расширенных связей между требованиями, расположенными на трех уровнях.

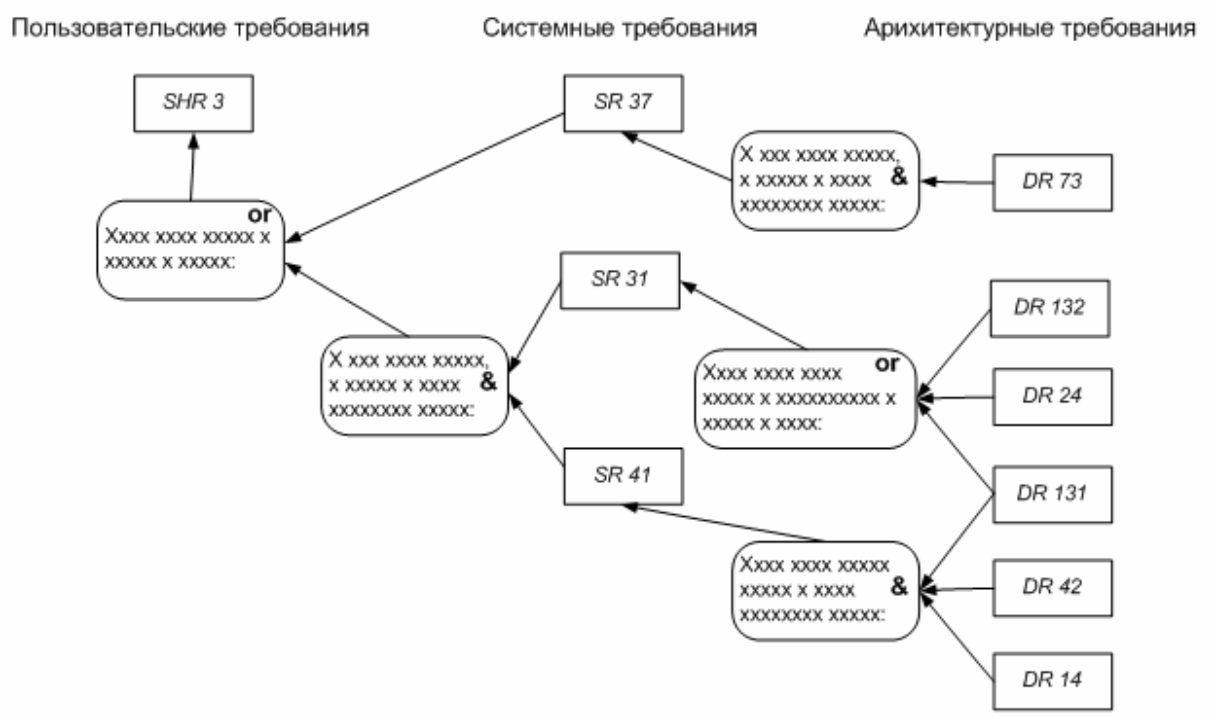


Рис. 7.7 Расширенные связи для нескольких уровней требований.

7.4 «Спуск» требований

В том случае, если по отношению к нескольким разным подсистемам или компонентам существует идентичное (тождественное) требование, то позиционирование аргумента удовлетоворения служит только для того, чтобы обозначить месторасположение этого требования, поскольку содержание самого аргумента уже не вызывает трудностей. Именно поэтому такой процесс часто называют «назначением» требования или «спуском».

При этом, если известно, что требование нижнего уровня напрямую (без изменения) «спущено» с верхнего уровня, то процесс внесения изменений в требования может быть сильно упрощен. Другими словами, все изменения в требовании верхнего уровня могут быть автоматически «спущены» на нижние уровни.

Небольшое и простое дополнение к расширенным связям позволяет отражать суть подобных операций. Для иллюстрации этого в дополнение к операторам «or» и «&» в аргументах удовлетворения может использоваться оператор «тождества», изображающийся символом «=».

На рисунке 7.8, который является развитием предыдущего, показан пример использования этого оператора.

Рис. 7.8 Спуск требования с использованием оператора «тождество».

7.5 Анализ связей

Каждый раз, когда требование анализируется и рецензируется, оно должно обязательно рассматриваться только вместе с его аргументом удовлетворения. Процесс анализа, базируясь на возможностях расширенных связей, может быть настроен таким образом, чтобы отображать только «нужную» информацию, например, только одно требование вместе с его аргументом удовлетворения, а также требованиями, которые из него вытекают.

На рис. 7.9 показана копия экрана специализированного программного обеспечения, использовавшегося в одном из проектов в военной отрасли. Пример иллюстрирует возможность видеть требования и аргументы удовлетворения, т.е. наблюдать только необходимую в данный момент информацию, чтобы оценить само требование и то, каким образом оно удовлетворяется.

Красные треугольники в нижней части экрана позволяют перемещаться к требованиям более низких уровней, а треугольник в верхней части предназначен для перехода на требования того же уровня. Т.е. совокупность знаков навигации позволяет проследить, таким образом, полный путь удовлетворения требования.

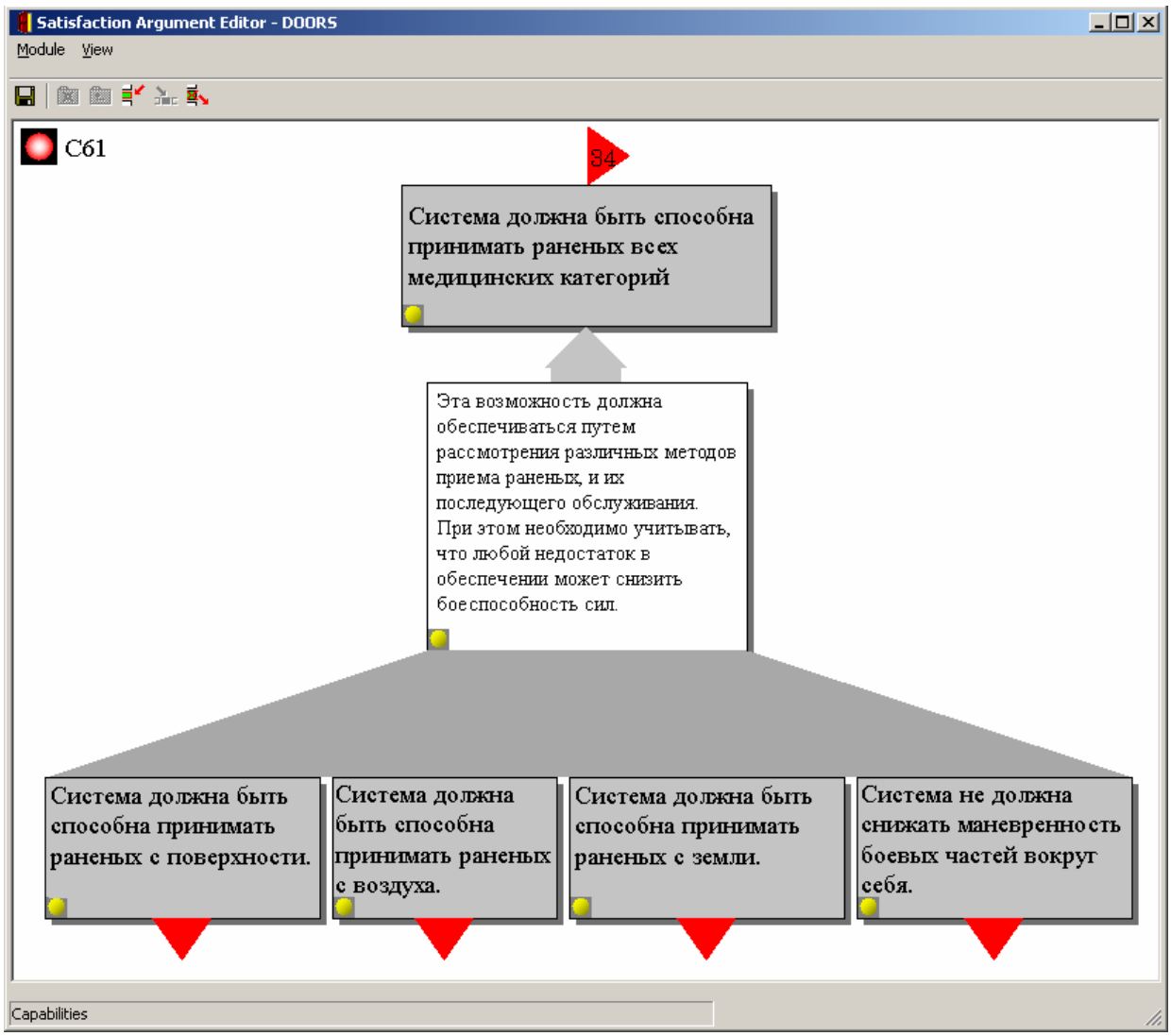


Рис. 7.9 Программное средство анализа аргументов удовлетворения.

7.6 Язык написания аргументов удовлетворения

При работе с аргументами удовлетоворения, было бы весьма полезно иметь – по аналогии с требованиями - унифицированный подход для формулировки выражения аргументов. Ключевым принципом, который можно принять здесь на вооружение, является правило начинать предложение с фразы: «*Это требование будет удовлетворяться ... <чем> ...*», что помогает сосредоточиться на сути аргумента удовлетворения и сформулировать его более корректно.

В то время, как требования должны быть «атомарными» (см. главу 4), аргументы удовлетворения не должны быть столь сильно ограничены. Однако, если формулировка

аргумента становится слишком сложной и «перегруженной», то лучше всего такой аргумент реструктуризировать, т.е. разбить на составляющие.

Если среди аргументов удовлетворения можно выделить типовые, то их лучше использовать для подготовки шаблонов аргументов, которые и использовать в последующем для большего эффекта.

7.7 Анализ расширенных связей

Присутствие аргументов удовлетворения в расширенных связях не в коей мере не препятствует возможности выполнения простейшего анализа влияния и анализа последствий, как это описано в главе 1. Напротив, аргументы удовлетворения являются важным ключом к пониманию сути такого влияния и помогают лучше понять характер связей между требованиями.

Более того, использование пропозициональных операторов (**&** и **or**) в аргументах удовлетворения дает дополнительную возможность для проведения анализа других типов. Так, например, структура связей может быть проанализирована на предмет определения числа степеней свободы, которые необходимы для достижения определенной цели.

Если в качестве примера взять структуру аргументов удовлетворения, изображенную на рис. 7.4, то пропозициональная структура для требования ***UR3*** может быть выражена следующим образом:

SR37 or (SR 31 & SR41)

Тогда, используя правила пропозициональной логики, эта запись, - не меняя ее смысла, - может быть преобразована в несколько альтернативных выражений, каждое из которых отражает свой собственный способ возможной реализации (удовлетворения) требования:

[SR37 & (not SR 31) & (not SR41)] или
[SR37 & SR 31 & (not SR41)] или
[SR37 & (not SR 31) & SR41] или
[SR37 & SR 31 & SR41] или
[(not SR37) & SR 31 & SR41]

Для простых сценариев возможность проведении такого анализа может показаться не очень уж и полезной, однако если представить себе более сложный случай с сотнями взаимодействующих требований, да еще и лежащих на нескольких уровнях, то полезность такого анализа возможных альтернатив становиться очевидной. Ведь такой подход позволяет не только выявить *различные* способы удовлетворения требований, но и находить конфликты, если в результате подобных логических построений не остается ни одного сценария, приводящего к достижению первоначальной цели.

7.8 Использование расширенных связей для тестирования

Метод расширенных связей может применяться в отношении связей любого типа. Хотя до сих пор речь, в основном, шла о связях типа *удовлетворение*, тем не менее этот метод

может также применяться и в отношении связей типа *проверка*. В этом случае термин «аргумент удовлетворения» может быть заменен термином «обоснование проверки» или «аргумент проверки». При этом все преимущества метода расширенных связей также будут проявляться и в области стратегии проверки.

7.9 Реализация метода расширенных связей

В этом разделе мы опишем два похода к реализации метода расширенных связей – одноуровневый и многоуровневый.

7.9.1 Одноуровневые расширенные связи

Данный подход проиллюстрирован на рис. 7.10.
Каждое требование верхнего уровня имеет единственную формулировку удовлетворения или стратегии в качестве атрибута. А несколько требований уровня ниже могут вытекать из него, образуя отношения «удовлетворяет» типа многие-ко-многим (стрелки связей показывают направление удовлетворения).
Для того, чтобы классифицировать этот аргумент как конъюнкцию (**&**) или дизъюнкцию (**or**), используется другой атрибут (на рисунке не показан).

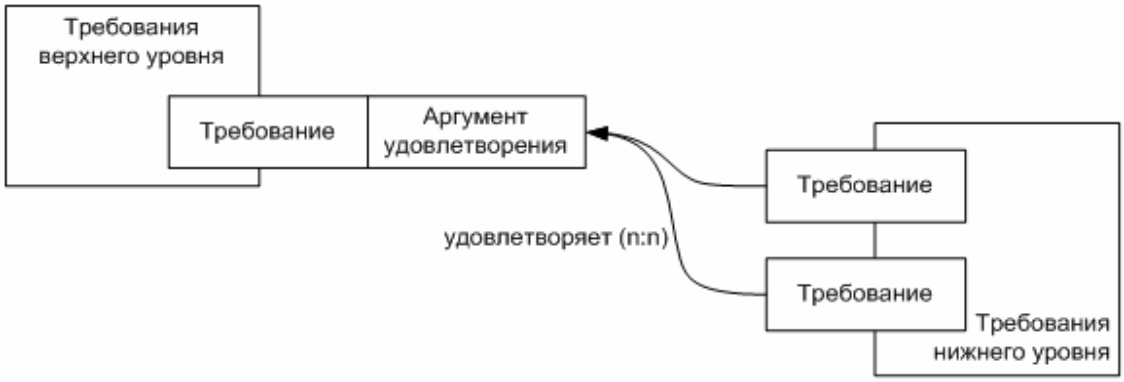


Рис. 7.10 Одноуровневая расширенная связь.

7.9.2 Многоуровневые расширенные связи

В данном случае аргумент удовлетворения сам по себе будет выглядеть как многоуровневая структура, состоящая из главного аргумента более высокого уровня и аргументов низкого уровня, которые призваны поддержать главный аргумент.
Главный аргумент «привязан» к требованию либо как атрибут, либо через отношение «устанавливает». А требования более низкого уровня, в свою очередь, связаны с аргументами низкого уровня связью типа «участвует», как это показано на рис. 7.11.

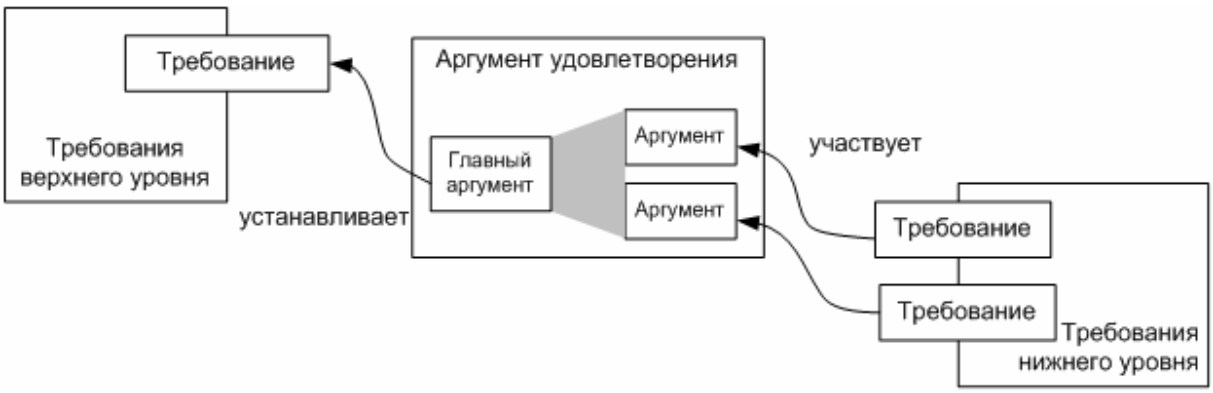


Рис. 7.11 Многоуровневая расширенная связь.

В большинстве систем общее количество уровней аргументов ограничено двумя. Верхний уровень – для главного аргумента, нижний - для аргументов, которые «расшифровывают» роли, выполняемые каждым требованием более низкого уровня в удовлетворении исходного требования.

7.10 Проектная документация системы

Проницательный читатель, наверняка, уже заметил, что идея аргументов удовлетворения весьма напоминает основную идею «сэндвича» управления требованиями, представленного на рис. 1.9, - только здесь «начинкой» между слоями являются аргументы удовлетворения. Далее – проще: если все аргументы удовлетворения собрать в единый документ, то все вместе они сформируют аналитическую и проектную документацию. Эта документация и будет являться связующим звеном между требованиями и моделированием.

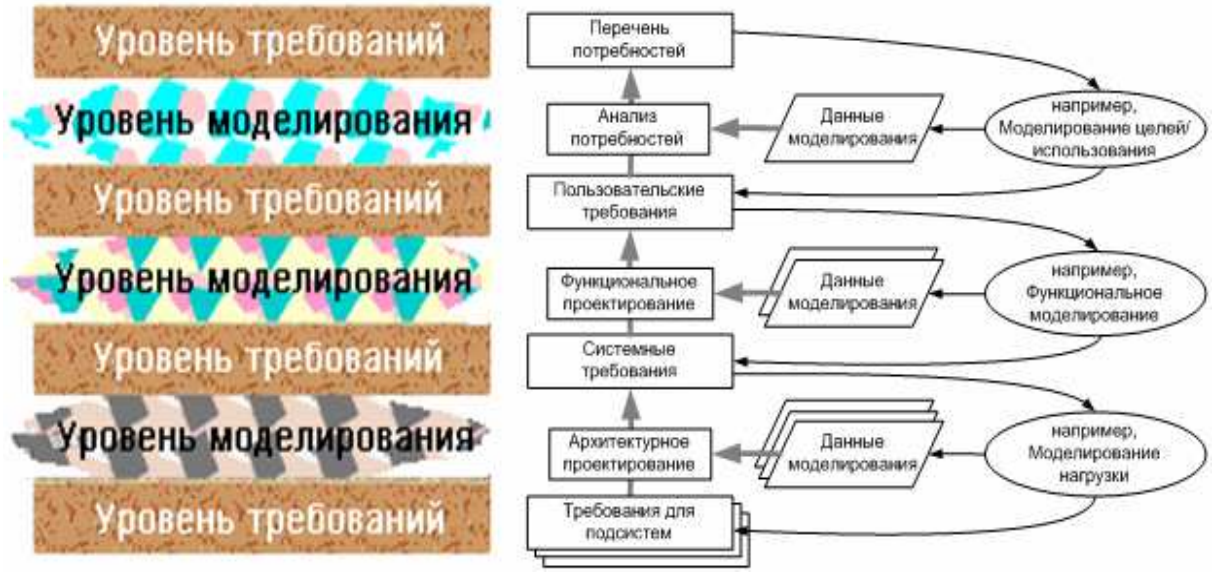


Рис. 7.12 Аналитическая и проектная документация.

Роль проектной документации состоит в том, чтобы агрегировать – в тестовой и графической формах – все те разрозненные части моделей и сценариев, которые расшифровывают, почему требований данного уровня являются необходимыми и достаточными для удовлетворения требований более высокого уровня. Документ будет содержать данные, полученные в результате моделирования, которые подтверждают обоснованность принятых проектных решений. При этом связи между уровнями требований фактически присутствуют и в этом проектном документе. Следовательно результаты моделирования также оказываются включенными в цепочку связей и могут вовлекаться в процесс анализа, в частности, в процесс анализа влияния.

Рис. 7.12 иллюстрирует данный подход. Между уровнями требований располагаются проектные документы. Результаты моделирования на каждом уровне формируют данные, на которые ссылается соответствующий проектный документ. Тонкие стрелки обозначают потоки информации, а толстые стрелки - связи (трассировку).

Давайте рассмотрим пример, поясняющий какого рода информация может включаться в проектный документ.

	1 Основные понятия
	Данный раздел будет содержать текстовое описание всех моделируемых сущностей. Каждая их них будет соответствовать понятию «класс» в модели UML.
	1.1 Единица багажа Термин ‘Единица багажа’ обозначает один предмет, сдаваемый в багаж.
	Каждый пассажир может иметь 0 или более единиц багажа.
	1.2 Учитываемая единица багажа Термин ‘Учитываемая единица багажа’ обозначает единицу багажа, для которой может быть однозначно определена ее принадлежность определенному пассажиру.
	1.3 Багажная квитанция Термин ‘Багажная квитанция’ обозначает средства, позволяющие пассажиру подтвердить принадлежность ему единицы багажа.
	1.3.1 Определяет Багажная квитанция предназначена для однозначной идентификации единицы багажа.
	1.4 Связи между основными понятиями

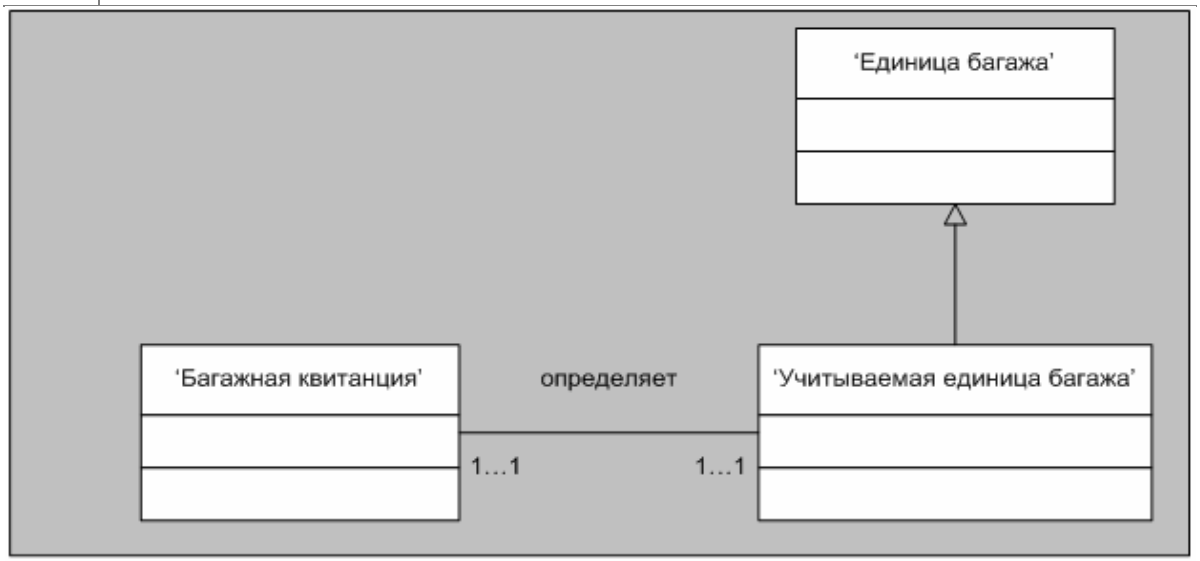


Рис. 7.13 Раздел «Основные понятия».

Последовательность рисунков этого раздела иллюстрирует выдержки из документа «анализ потребностей», который описывает модель системы регистрации багажа в области проблем. Модель, если можно так выразится, располагается «между» документом, описывающим «перечень потребностей» и документом, отражающим «пользовательские требования» (см. рис. 7.12), и использует графическую нотацию и язык моделирования UML 2 для отображения анализа в визуальной форме.

Перечислим основные виды информации, которые включаются в документ:

Основные понятия. Для отображения основных понятий из области проблем и отношений между ними используется диаграмма классов UML. Каждое понятие изображается в виде класса UML, а каждое отношение - в виде ассоциации UML. И классы и ассоциации являются компонентами документа и отображаются в нем в текстовой форме. На рис. 7.13 показан пример системы регистрации багажа. Значок слева от параграфа обозначает, что эта часть документа соответствует определенному объекту в UML модели.

Заинтересованные стороны. Данный раздел документа перечисляет все заинтересованные стороны, которые были определены в результате анализа, и содержит диаграмму классов, показывающую отношения между ними. Рис. 7.14 иллюстрирует пример системы с двумя заинтересованными сторонами и одним отношением между ними.

3 Заинтересованные стороны

Данный раздел должен содержать текстовое описание каждой из заинтересованных сторон. Каждая из них должна соответствовать понятию «актер» в модели UML.

3.1 Пассажир

Пассажир - это любой человек, собирающийся полететь на самолете.

3.2 Семья

Семья - это группа пассажиров, путешествующих вместе.
Они могут иметь общий багаж, а также желание сидеть рядом.

3.3 Отношения между заинтересованными сторонами

Дополнительная диаграмма классов может показывать отношения между заинтересованными сторонами, в случае, если они существуют.

Рис. 7.14 Раздел «Заинтересованные стороны».

Статическое окружение. Целью этого раздела (рис. 7.15) является идентификация окружения, в котором существует система регистрации багажа. На диаграмме классов система регистрации багажа, наряду с окружающими и составляющими ее внутренними системами, изображается в виде класса. Для моделирования отношений между этими системами используются ассоциации и включения. Как уже отмечалось выше, каждый класс и ассоциация появляются в документе в виде текстового описания.

	3 Окружение
	Данный раздел должен начинаться с диаграммы классов, описывающей основной контекст разрабатываемой системы.
	3.1 Система регистрации багажа *Данный раздел определяет и описывает разрабатываемую систему.*
	3.2 Система управления потоками багажа *Эта система является внутренней по отношению к разрабатываемой. (Концепция система-в-системе.)* *Эта система должна быть представлена в виде «класса», для того чтобы ее можно было изобразить на архитектурных диаграммах, но мы идентифицируем ее как «актер».*
	3.3 Система транспортировки багажа *Это система того же уровня иерархии - внутренняя по отношению к разрабатываемой. Мы отобразим ее как «класс», но со стереотипом «актер».*
	3.4 Система транспортировки пассажиров *Это система того же уровня иерархии - внутренняя по отношению к разрабатываемой. Мы отобразим ее как «класс», но со стереотипом «актер».*
	3.5 Система востребования (получения) багажа *Это система того же уровня иерархии - внутренняя по отношению к разрабатываемой. Мы отобразим ее как «класс», но со стереотипом «актер».*
	3.6 Система хранения багажа *Это система того же уровня иерархии - внутренняя по отношению к разрабатываемой. Мы отобразим ее как «класс», но со стереотипом «актер».*
	3.7 Отношения с окружающими системами

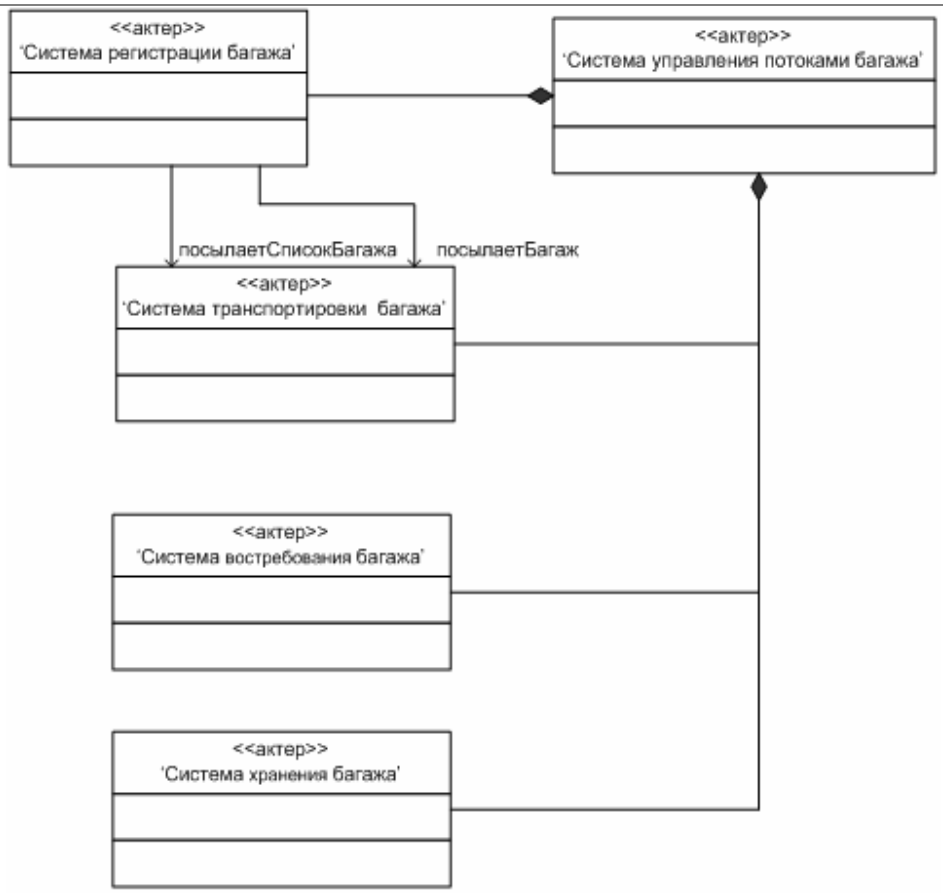


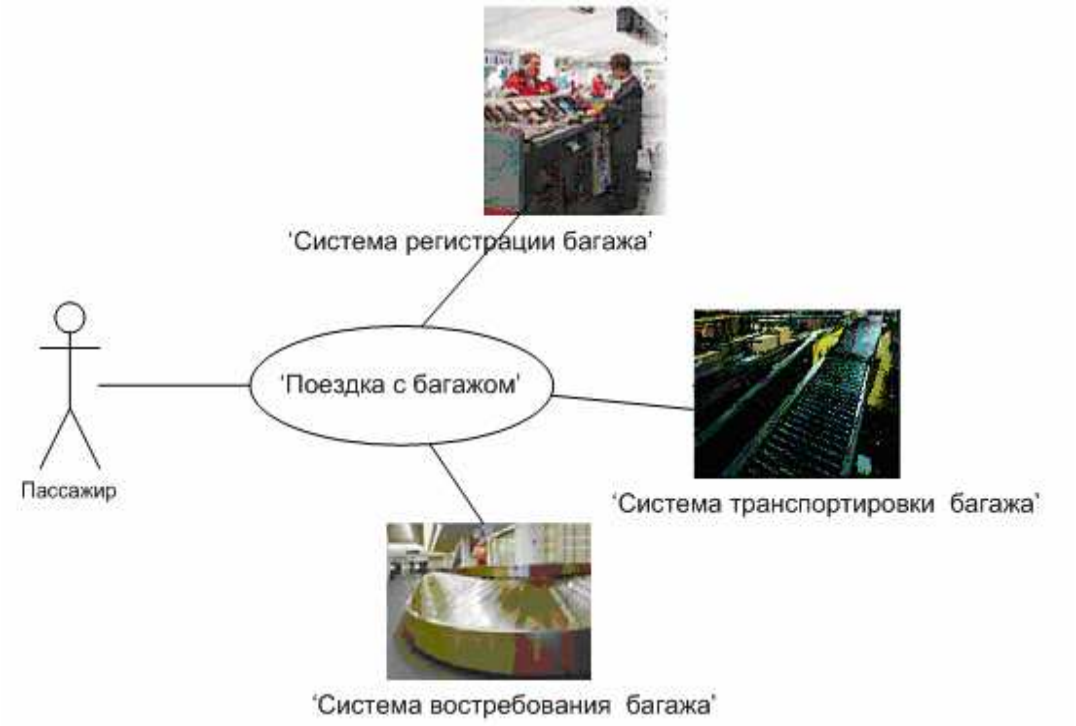
Рис. 7.15 Раздел «Окружение».

4 Функциональность

Данный раздел содержитосновные прецеденты (верхнего уровня).

Поездка с багажом
Общее описание прецедента.

Прецедент



4.1.2 Поездка с багажом: нормальный ход событий

Описание нормального хода событий для этого прецедента.

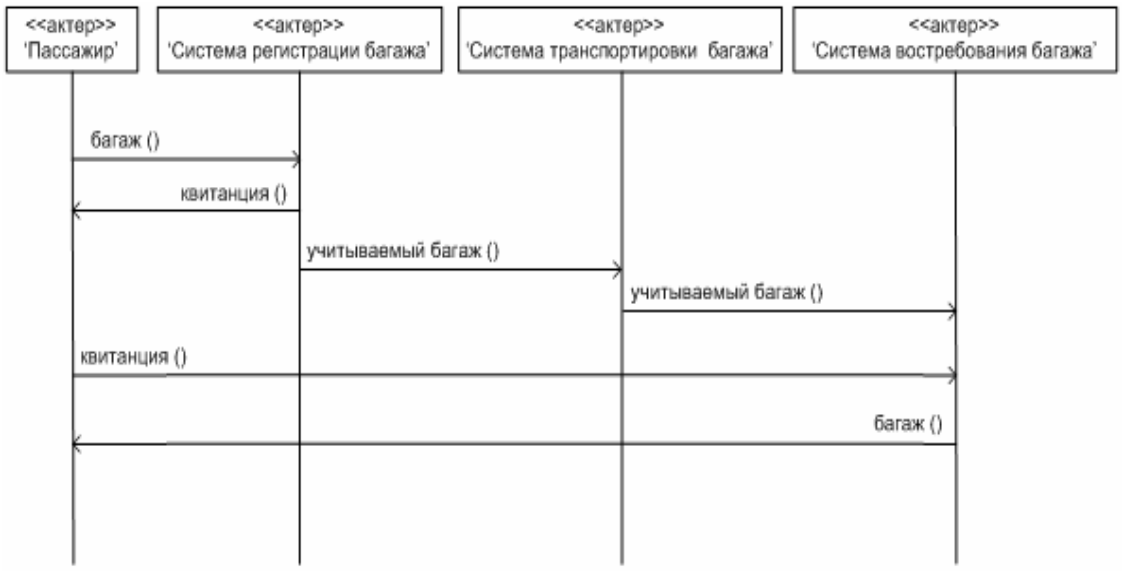


Рис. 7.16 Раздел «Функциональность».

• *Функциональность*. В этом разделе описываются основные (верхнего уровня) прецеденты для системы. Для этого используется серия диаграмм прецедентов, каждая из которых имеет одну или более диаграмм последовательностей. На рис. 7.16 показана всего одна диаграмма прецедента и соответствующая ей диаграмма последовательностей, иллюстрирующие нормальный ход развития событий сценария. Диаграмма последовательностей отражает взаимодействие между заинтересованными сторонами (некоторые из которых являются внешними системами) и рассматриваемой системой (системой регистрации багажа), что позволяет определить границы системы, суть ее работы и внешние интерфейсы.

1. Обоснования

1.1 Уменьшение времени регистрации

[SoN-9]
Для одного и того же количества багажа пассажиры будут иметь возможность в среднем регистрировать багаж в два раза быстрее.
В настоящее время среднее время регистрации одной единицы багажа составляет 80 секунд.

Данная цель будет достигаться тем, что система регистрации багажа будет иметь достаточную производительность в точке приемки багажа. Регистрация пассажира и регистрация багажа будут разделены. Расчетное время регистрации единицы багажа составляет 25 секунд. Данный показатель может быть достигнут следующими способами:

- регистрация пассажира и регистрация багажа являются отдельными событиями в моделируемых сценариях;
- требования производительности предъявляются к системе регистрации багажа.

[PA-233] Диаграмма последовательностей UML «Поездка с багажом: нормальный ход событий»

[SHR-3] В аэропорту отправления пассажир должен иметь возможность зарегистрировать единицу багажа не позже, чем через 25 секунд, с того момента, как багаж был помещен на ленту транспортера.

1.2 Усиление безопасности

[SoN-8]
Пассажиры будут иметь возможность забирать только тот багаж, который они сами зарегистрировали и сдали при вылете.

Данная цель будет достигаться использованием «уникальных» багажных квитанции для пассажиров, регистрирующих свой багаж при вылете. Это позволит однозначно соотнести багажную квитанцию и багаж и заставит пассажиров предъявлять багажные квитанции «Системе востребования (получения) багажа» в точке прилета. Данная стратегия достигается с помощью следующего:

- соблюдается строгое разграничение между просто багажом и отслеживаемым багажом;
- каждая единица отслеживаемого багажа имеет уникальный номер (квитанцию);
- предъявление пассажиром багажной квитанции происходит в месте выдачи (получения) багажа.

Единица багажа
[PA-3] Термин ‘Единица багажа’ обозначает один предмет, сдаваемый в багаж.

Отслеживаемая единица багажа
[PA-6] Термин ‘Отслеживаемый багаж’ обозначает единицу багажа, принадлежность которой может быть однозначно соотнесена с определенным пассажиром.

... определяет
[PA-263] «Багажная квитанция» однозначно определяет «единицу багажа».
[PA-233] Диаграмма последовательностей UML «Поездка с багажом: нормальный ход событий»

[SHR-3] В аэропорту прибытия пассажир должен иметь возможность забрать багаж, который он сдал в в аэропорту вылета.

Рис. 7.17 Раздел «Обоснования».

- *Обоснования*. Данный раздел обобщает результаты анализа и моделирования и дает развернутые обоснования тому, каким образом потребности будут удовлетворяться возможностями системы. Одним из способов представления этой информации является использование «аргумента удовлетворения» для каждого из входящих требований (потребностей). Это именно здесь устанавливается связь, идущая ***к*** требованиям высокого уровня и идущая ***от*** требований нижнего уровня. В сущности, аргумент удовлетворения призван объяснить здесь, каким образом формулировка потребности декомпозируется на утверждения возможностей (см. рис. 7.17).

Левая колонка приведенной таблицы содержит формулировку потребности, средняя колонка содержит формулировку логического обоснования, правая колонка - подтверждение обоснованности выбранного решения в виде ссылок на модель и требования, на основании которых эта обоснованность и выводится. Табличное представление, в сущности, является еще одним «сэндвичем»: два слоя требований, а между ними слой, подтверждающий обоснованность выбранных решений.

При использовании эффективного программного инструмента для управления требованиями, данная таблица может быть сформирована автоматически на основании связей между уровнями требований.

7.11 Параметры и метрики связей

Поскольку связи и их анализ занимают ключевое место при разработке и управлении требованиями, было бы интересно рассмотреть возможность использования метрических характеристик для оценки потоков требований и связей между ними.
Если, начав с верхнего уровня, опускаться вглубь различных слоев требований, двигаясь по связи типа «удовлетворение», то можно выделить три основных параметра трассировки, которые могут быть охарактеризованы следующим образом:

- ***Широта:*** насколько хорошо связи данного уровня «охватывают» требования верхнего (или нижнего) уровней треб ований?
- ***Глубина:*** насколько глубоко вниз (или высоко вверх) через уровни продолжается выбранная связь?
- ***Нарастание:*** насколько широко «разрастается» связь через уровни?

Для того, чтобы определить, какой именно и как каждый из этих трех параметров может быть полезен с точки зрения оценки процесса разработки и управления требованиями, необходимо прежде всего размежевать два типа метрик:

- ***Фазовые метрики:*** это оценки, применимые к одной из фаз процесса разработки требований (*напр.*, только к фазе разработки системных требований).
- ***Глобальные метрики:*** это оценки, которые могут характеризовать сразу несколько фаз процесса разработки требований.

Давайте рассмотрим теперь более подробно все три параметра, а также соотношение между ними.

7.11.1 Широта

Этот параметр характеризует «покрытие» (степень охвата) и по сути является фазовой метрикой. Как обсуждалось в главе 1, «покрытие» может использоваться для количественной оценки хода работ, в рамках которых создаются связи между требованиями. Этот параметр оценки соотносится с одним (каждым) уровнем требований и показывает степень покрытия для уровня находящегося или выше, или ниже (или «рядом», если рассматривается степень покрытия требований этого уровня тестами).
Другими словами, - если любое из требований, например, верхнего уровня удовлетворяется хотя бы одним или несколькими требованиями нижнего уровня, то параметр широты может быть охарактризован своим максимальным значением; если же набор требований нижнего уровня не удовлетворяет всех требований верхнего уровня, то стоит говорить о некоем промежуточном значении параметра широты.

7.11.2 Глубина

Глубина показывает на какое количество уровней вверх (или вниз) распространяется действие связи от данного конкретного уровня. Таким образом, параметр глубины является глобальной метрикой. Этот параметр может быть применим в ситуации, когда ищутся источники для требования, находящегося на самом нижнем уровне.
Этот параметр поможет оценить сколько требований, пройдя через несколько уровней, полностью «спустилось» из пользовательских требований до уровня, например, подсистем; или как много из них «спустилось» еще ниже и появилось в результате выработки проектных решений?

7.11.3 Нарастание

Нарастание - это наиболее интересный параметр. Он связан с оценкой потенциального влияния изменений. Параметр характеризует то, сколько требований на нижних уровнях связано с одним требованием на самом верхнем уровне.
Рассмотрим рис. 7.18, на котором приведены четыре характерных примера.

В том простом случае (a), когда одно требование высшего уровня удовлетворяется одним требованием уровня ниже, можно с очевидностью утверждать, что параметр нарастания равен 1. В случае же (b), когда одно требование удовлетворяется, например, шестью требованиями, параметр нарастания будет равен уже 6.
Какую разницу между двумя требованиями характеризует в данном случае параметр ?
Вполне возможно, что:

- требование верхнего уровня (b) плохо сформулировано и, видимо, нуждается в разбиении на несколько отдельных декомпозиционных требований
- требование верхнего уровня (b) более сложное по сути, чем требование (a), и, следовательно, ему нужно уделить больше внимания, расписав подробней;
- изменения в требовании верхнего уровня (b), будут иметь гораздо бóльшие последствия, чем изменения в требовании (a), и, следовательно, нужно более осторожно и внимательно относиться к его потенциальным изменениям.

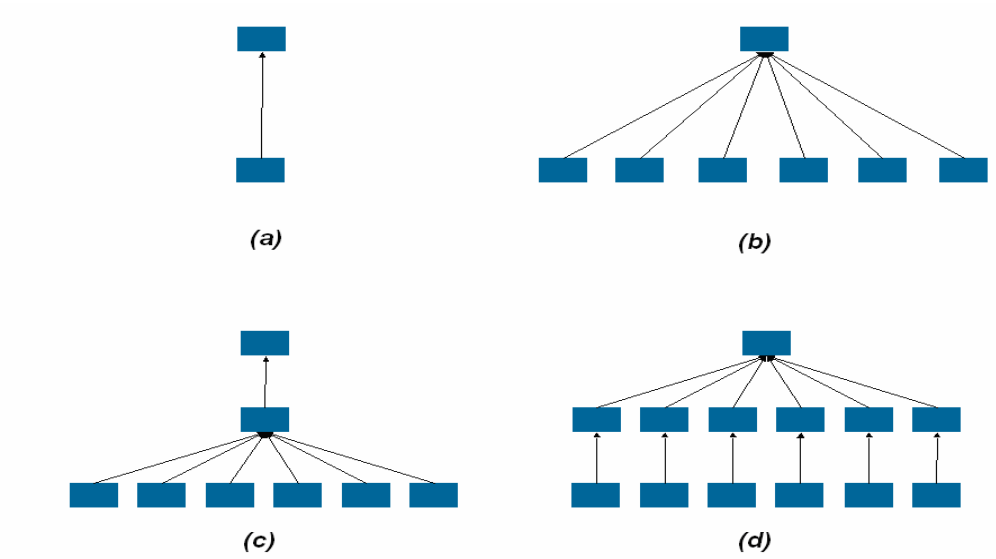


Рис. 7.18 Широта связей.

Конечно же, это явное расхождение в значении фактора нарастания на данном уровне можно компенсировать, спустившись еще на один уровень ниже. Это иллюстрируется примерами (c) и (d), где значения фактора нарастания уже одинаковы на втором нижнем уровне.
Итак, какие выводы можно сделать из этих примеров?
Возможно, что:

- требование верхнего уровня (c) было слишком абстрактно и требовало «расшифровки»;
- требования среднего уровня (d) были излишне подробны для этого уровня.

Следует заметить, что только после продолжительного опыта работы с этим параметром в организациях, которые разрабатывают системы определенного типа, удается добиться получения желаемого значения фактора нарастания требований между уровнями. Более привычным использованием этого параметра является применение его как средства для проверки пропорциональности прироста требований, что позволяет идентифицировать потенциально неконтролируемые требования или несогласованности в процессе разработки различных уровней требований.

7.11.4 Равномерность

Целью использования метрик является анализ распределения значений фактора нарастания для индивидуальных требований между двумя заданными уровнями и последующая экспертиза тех требований, которые располагаются во внешних квартилях[8].

[8] *прим. переводчика*. Квартиль – граница на шкале измеряемого (тестируемого) параметра, отделяющая 25% испытуемых от общей выборки. Различаются три квартиля: Q1 - первые 25%, Q2 - 50% (медиана), Q3 - 75%.

Другими словами, основной задачей является выявление требований, имеющих «ненормально» высокий или «ненормально» низкий фактор нарастания. Такие требования необходимо подвергнуть дополнительной тщательной проверке.

На рис. 7.19 показано, как может выглядеть достаточно типичное распределение значений фактора нарастания.

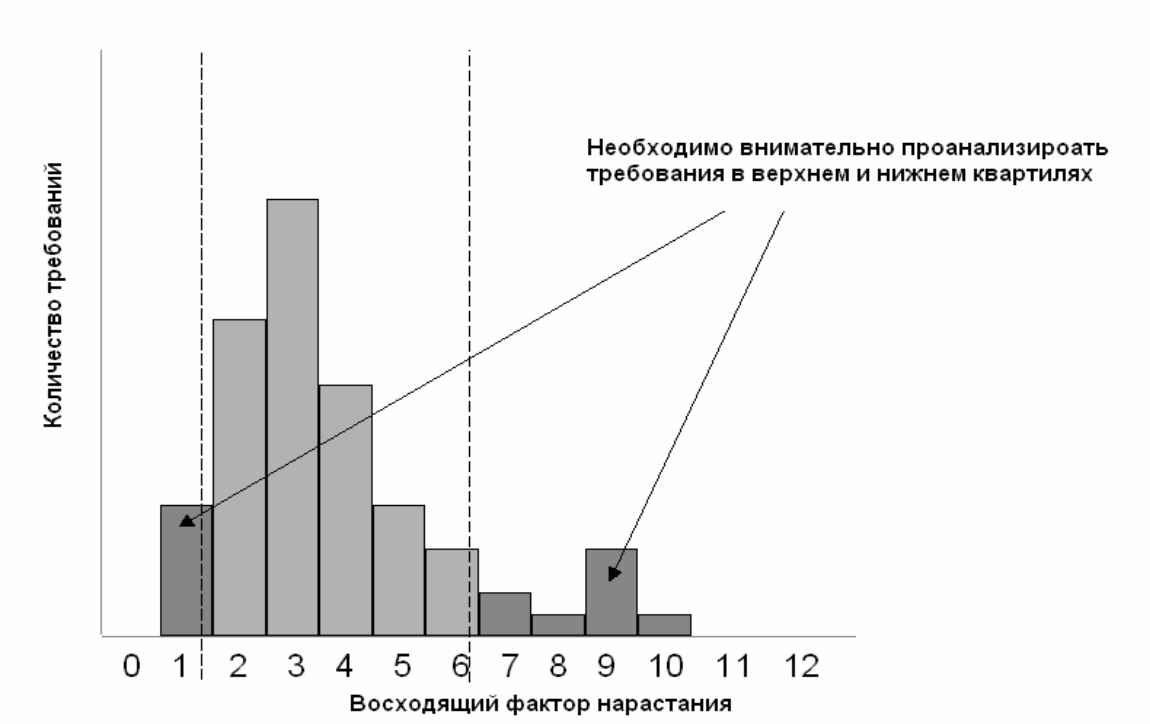


Рис. 7.19 Частотное распределение фактора нарастания требований.

По горизонтальной оси располагается фактор нарастания, а высота столбика по вертикальной оси соответствует количеству требований, которые имеют данное значение фактора. Как видно из рисунка, для большинства требований фактор нарастания лежит в диапазоне от 2 до 6, и только лишь небольшое количество требований имеют фактор нарастания либо равный 1, либо больший, чем 6. Последние явно нуждаются в более пристальном и особом внимании.

Следует заметить, что все приведенные выше рассуждения относились к случаю, когда нарастание «идет вниз», т.е. мы выясняли какое количество требований «вытекает» из определенного требования (нисходящий фактор нарастания). Однако, это не единственно возможное направление анализа. Вполне возможно провести анализ и в обратном направлении, т.е. выяснить какое количество требований «втекает» в данное требование.

Давайте рассмотрим рис. 7.20, не забывая о том, что связи между требованиями имеют характер «многие-ко-многим».

Два требования нижнего уровня (обозначены, как 2 и 3) имеют более одной исходящий связи, т.е. каждое из них «втекает» в более, чем одно требование высокого уровня. Что особенного можно сказать об этих требованиях? Возможно, эти требования более важны (критичны), чем остальные, поскольку от них зависит удовлетворение сразу нескольких требований выше и, следовательно, им нужно уделить больше внимания.

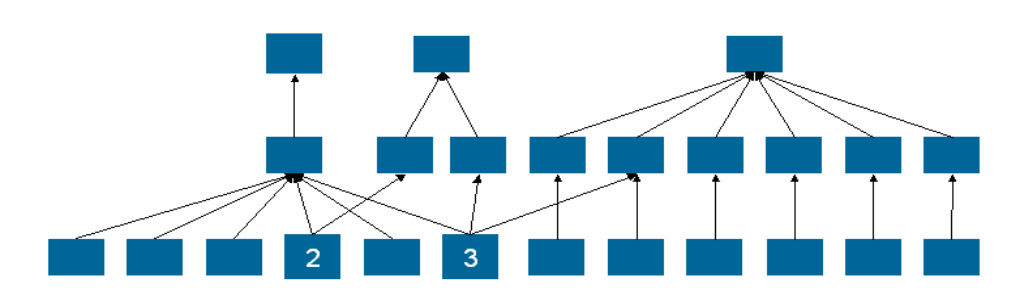


Рис. 7.20 Критичность требований.

Для выявления такого рода требований и может использоваться анализ распределения восходящего фактора нарастания. Рис. 7.21 демонстрирует типичное распределение значений фактора нарастания для этого случая.

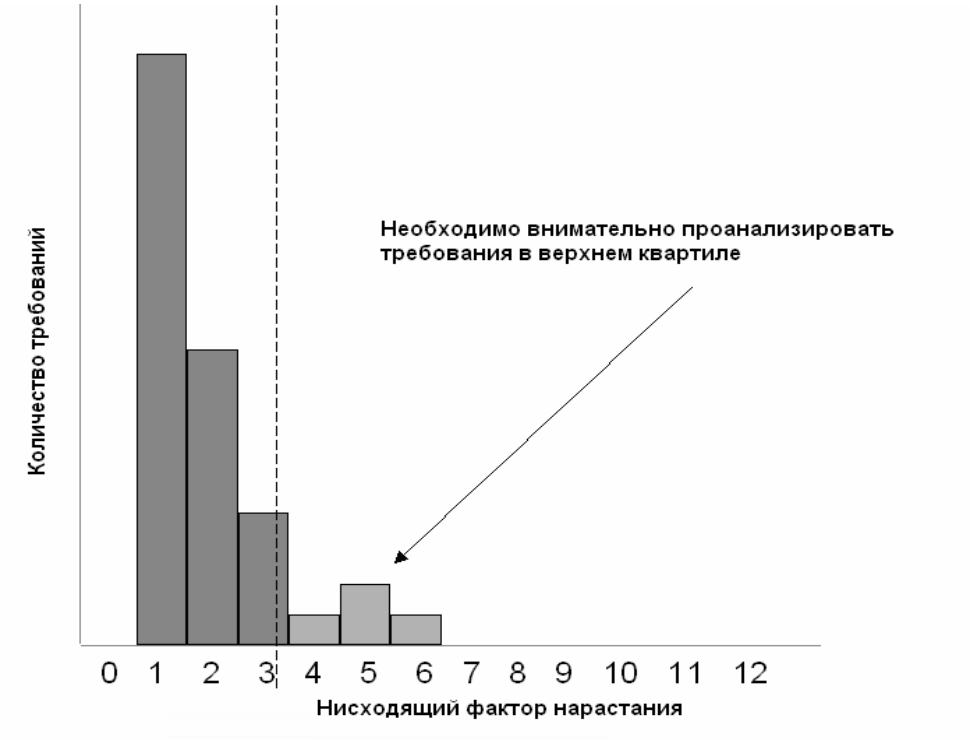


Рис. 7.21 Частотное распределение критичности требований.

7.11.5 Влияние изменения

Управление изменениями является, скорей всего, самой сложной задачей в дисциплине разработки требований. Основой процесс и его информационная модель, рассмотренные в главе 2, поясняют как преимущества наличия связей и возможности их анализа, помогают определить потенциальное влияние изменения.

После того, как в отношении данного требования появляется запрос на изменение (change request), все остальные требования, связанные с ним, должны автоматически становиться «подозрительными» (или сомнительными - *suspect*) до тех пор, пока специалисты-проектировщики не проанализирует реальное влияние изменения.

Очевидно, что одно единственное изменение, вносимое в требование, может вдруг вызвать целую лавину потенциальных изменений в системе. В этой ситуации возможность контролировать ход этих изменений, оценивать степень их влияния на всю систему, возможность оценивать ресурсы, необходмые для выполнения работ по реализации изменений, становиться жизненно важной.

Даже на простом примере рис. 7.22 нетрудно заметить насколько сложной и запутанной может быть воздействие на систему потенциального изменения.

Первичный запрос на изменение формулируется для одного требования, которое находится на самом верхнем уровне.

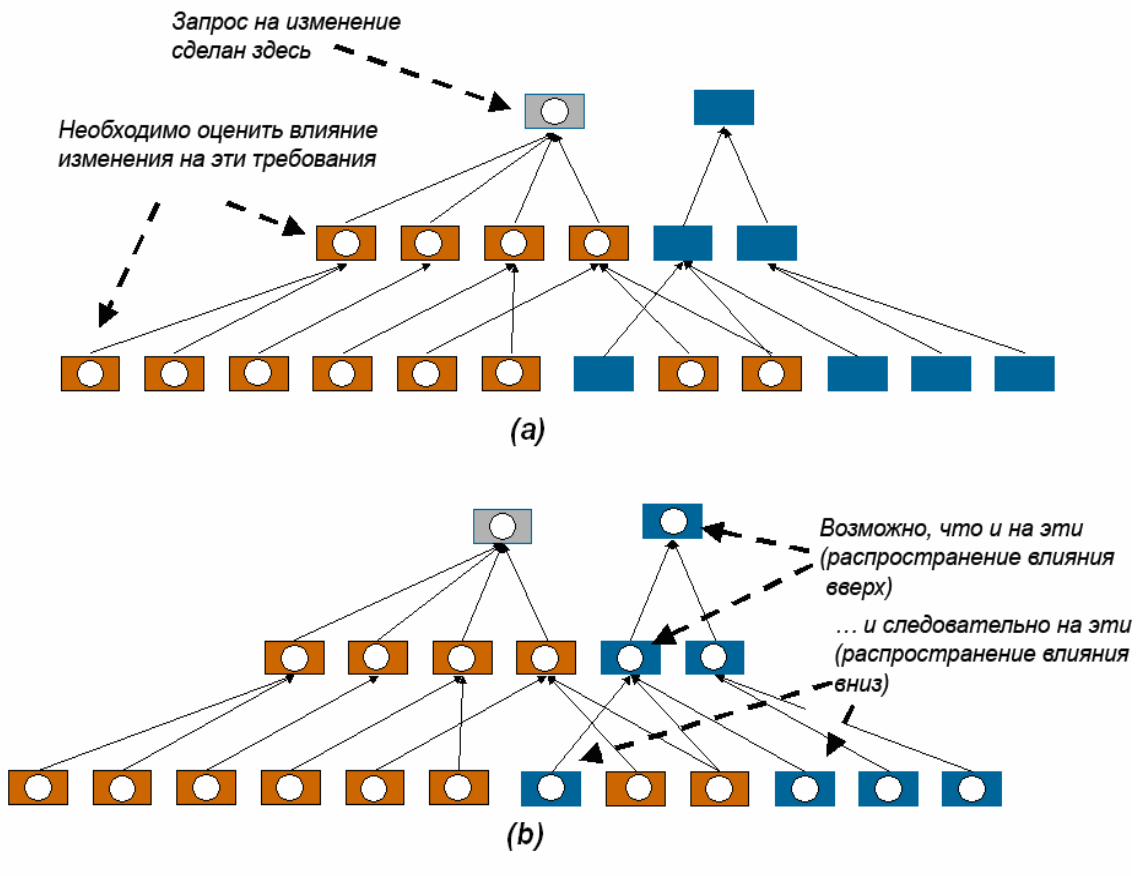


Рис. 7.22 Потенциальные изменения, вызванные запросом на изменение.

Часть (a) показывает возможное влияние изменения, которое выявляется с помощью анализа связей по направлению вниз (требования, которые необходимо проанализировать на предмет влияния на них этого изменения, показаны прямоугольниками с белым кружками). Часть (b) показывает вероятное влияние изменения, которое выявляется с

помощью анализа связей по направлению вверх. Это влияние проявляется уже как результат потенциальных изменений требований на нижних уровнях.
Таким образом, вначале необходимо оценить последствия первичного изменения на требования нижнего уровня (сверху-вниз), а затем уже пройтись в обратном направлении (снизу-вверх). Ведь очевидно, что после внесения изменений в требования нижнего уровня, они могут престать удовлетворять связанные с ними требования верхнего уровня, что, в свою очередь, потребует внесения ряда новых дополнительных изменений, а также согласований требований на верхних уровнях.
Очень быстро становиться ясно, что ***все*** требования в приведенном примере являются «подозрительными» только из-за одного единственного запроса на изменение!

Конечно, после того, как аналитик начнет внимательно оценивать реальное влияние вносимого изменения, часть связанных требований может покинуть разряд «подозрительных», что, к счастью, уменьшит общий поток потенциальных изменений; а в некоторых случаях даже очень существенно.

Текущее положение процесса оценки изменения может быть охарактеризовано количеством требований, все еще имеющих статус «подозрительное». В самом начале, после запроса на изменение, все требования, связанные с данным, помечаются как «подозрительные». Затем, в процессе анализа потенциального влияния количество подозрительных требований будет постепенно уменьшаться, т.е. у каждого требования, влияние изменения на которое не установлено, признак «подозрительное» будет сбрасываться, что, в свою очередь, может вызывать каскад сбросов этого признака и у других требований, связанных с данным.
Рис. 7.23 демонстрирует вышесказанное, - после формирования запроса на изменение, количество подозрительных требований резко возрастает, образуя пик, а затем – по мере анализа - это количество идет на спад.

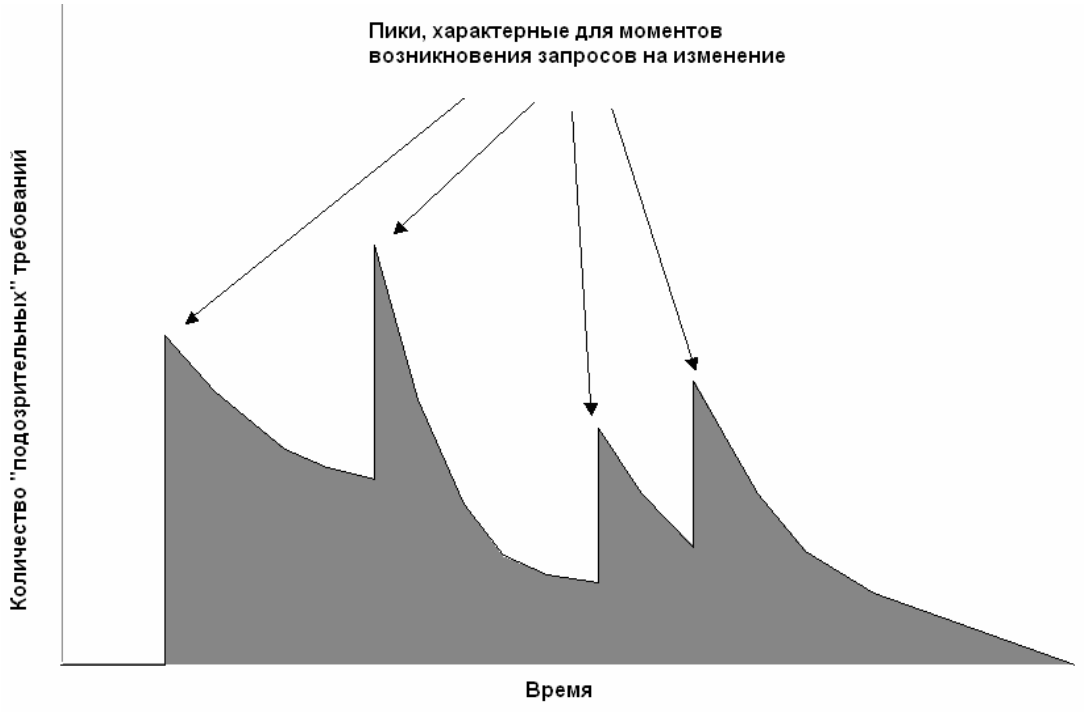


Рис. 7.23 Обработка запросов на изменение.

Следует особо подчеркнуть, что ранее мы рассматривали изменения, которые «распространяются» через *существующие* связи, но никак не касались последних. Однако на практике, потенциальные изменения могут вызывать необходимость либо появления новых, либо удаления существующих связей. Вот почему любое изменение, которое относится к связи, должно автоматически вызывать установку статуса «подозрительное» для требований на обоих концах этой связи.

7.12 Заключение

В дополнение к преимуществам, которые привносит возможность организации элементарных связей, о чем уже говорилось в разделе 1.5, метод расширенных связей позволяет более четко и ясно обосновать удовлетворение требований. Это обоснование выполняется с помощью аргументов удовлетворения, располагающихся на связях между требованиями. Использование аргументов значительно повышают уверенность разработчиков в правильности выбранного пути.

Несомненно, применение аргументов удовлетворения требует существенных дополнительных затрат, особенно для больших систем, требования для которых могут исчисляться сотнями и тысячами.

Для проекта железнодорожной сети, о которой мы упоминали, было написано около 500 аргументов удовлетворения, которые служили для разбиения требований высокого уровня на требования для подсистем. В течение почти трех лет команда аналитиков, включающая в разное время от 2 до 5 человек, занималась поддержкой этой информации.

Опыт однако доказал, что эти затраты многократно оправдали себя впоследствии. Возможность аргументировано продемонстрировать организациям, финансирующим проект, то, каким образом высокоуровневые цели заказчиков будут удовлетворены с помощью конкретных технических решений, явилась основой для «продажи» идей в проекте железнодорожной сети.

Очевидно, что связи и возможность их анализа являются богатым источником параметров и метрик, которые могут использоваться для измерения хода работ по проекту. Формализация связей и их периодический анализ дают возможность контролировать прогресс выполнения проекта.

8 Аспекты управления разработкой требований

Теория утверждает, что разницы между теорией и практикой не существует. Практика утверждает обратное.

Йоги Берра (Yogi Berra),
бейсболист, 1925 – н.в.

8.1 Введение в управление

Управление процессом разработки требований мало чем отличается от управления любыми другими процессами. Прежде, чем начать любой проект, необходимо понять, что именно должно быть получено в итоге. Необходимо представить характер работ, которые нужно будет выполнять. Также необходимо понять, будут ли существовать какие-либо взаимосвязи между отдельными задачами, например, одна из задач может быть начата только после завершения другой. И, наконец, необходимо знать какими профессиональными навыками должны обладать люди, которые будут выполнять эти задачи.

При составлении плана предстоящих работ, хорошей практикой является попытка заострить внимание именно на результате, который должен быть получен после выполнения каждой задачи. Поскольку результат может быть нагляден и измерим, то это обеспечит реальное подтверждение успешного выполнения задачи.

Базируясь на всей этой информации, мы можем создать план, который будет содержать информацию о том, какие задачи должны быть выполнены, кто будет выполнять эти задачи и в какие сроки эти они должны быть завершены. После этого может начаться реализация разработанного плана, а руководитель будет контролировать ход выполнения задач в соответствии с ним.

Если представить себе идеальный мир, в котором все происходит согласно плану, то наш проект должен безошибочно идти правильным путем, чтобы к назначенной конечной дате был бы получен финальный результат.

В реальной жизни все сильно отличается от идеала. Во-первых, очень тяжело дать точную оценку времени и трудозатрат, которые необходимы для выполнения задачи (за редким исключением той ситуации, когда руководитель проекта имеет большой опыт в управлении аналогичными задачами). Во-вторых, в процессе работы над проектом могут выявиться такие трудности, которые никак нельзя было предвидеть вначале (например, часть плана может быть составлена из расчета, что ключевой сотрудник будет выполнять ряд важных задач в определенный период времени, но по ряду причин оказывается, что этот человек в данное время недоступен).

Подобные ситуации зачастую приводят к отклонению от первоначального плана и требуют его пересмотра. Как только план пересмотрен и утвержден, все может повторяться заново. Постоянные корректировки плана приводят к тому, что стоимость проекта увеличивается, сроки реализации откладываются и все постепенно и неизбежно идет к

провалу. Альтернативным может быть вариант, при котором сроки окончания этапов и их стоимость сохраняются прежними, но зато уменьшается объем поставленных задач и выполняемых работ. Такой подход в некоторых ситуациях дает положительный результат. Например, если компании принципиально важно выйти на рынок с новым продуктом к определенной дате (в случае жесткой конкуренции) и при этом не выйти за рамки определенного бюджета (потому что компания не может позволить себе потратить больше), то тот предполагаемый функционал, который должен был бы иметь продукт, может быть уменьшен (необходим лишь определенный набор возможностей для того, чтобы новый продукт хотя бы чем-то отличался от предыдущего). Это может служить примером того, как экономика является главной движущей силой проекта.

Важно осознавать, что любой проект характеризуется тремя факторами:

- возможности создаваемого продукта;
- стоимость работ;
- сроки завершения.

Эти три фактора очень тесно связаны между собой (см. рис. 8.1). Любое изменение хотя бы одного из этих факторов приводит к изменению как минимум еще одного фактора. Рисунок также демонстрирует тот важный момент, что для продвижения проекта необходимо постоянно принимать решения. Каждое принимаемое решение заново позиционирует проект относительно этих трех фундаментальных факторов.

В своих снах и мечтах любой руководитель проекта видит, как каждое его решение значительно улучшает возможности продукта, одновременно снижая затраты и время на его разработку. Несмотря на свою практическую неосуществимость, эта идея имеет весьма широкое распространение.

Рис. 8.1 Взаимосвязь возможностей продукта, стоимости и сроков разработки.

8.2 Проблемы управления процессом разработки требований

В данном разделе мы познакомимся с теми специфическими проблемами, которые делают управление процессом разработки требований, несколько более сложным по сравнению с управлением любыми другими процессами.

Первой проблемой является то, что не так уж много людей владеют достаточным опытом управления требованиями. Это, в свою очередь, связано с тем, что лишь очень немногие организации имеют хорошо поставленный внутренний процесс управления требованиями. В результате люди, сталкивающиеся на проекте с необходимостью работать с требованиями, оказываются просто не готовы к этому. В такой ситуации дать оценку планируемого времени и трудозатрат невероятно трудно, поскольку одним из основных элементов хорошей оценки как раз и является соответствующий значительный опыт. Следовательно, в этом случае все выглядит не очень хорошо уже с самого начала. Тут уместно вспомнить шутку, когда один человек спрашивает другого о том, как ему попасть в определенное место, и в качестве ответа получает: «На вашем месте я бы не стал отсюда добираться туда»!

Как следствие из вышесказанного формируется даже более существенный вывод: если люди имеют небольшой опыт в управлении требованиями, они могут просто даже и не догадываться о том, какую работу необходимо запланировать для разработки требований. Поэтому в предыдущих главах был достаточно подробно описан перечень задач, которые необходимо выполнить в процессе разработки требований для систем различных типов.

Второй проблемой является то, что многие люди не отличают пользовательские требования от системных требований. Далее, они не понимают разницы между системными требованиями и системными спецификациями. Другими словами, они сразу начинают определять детальное техническое решение, вместо того, чтобы вначале разработать требования, не зависящие от реализации. Эту тему мы также неоднократно обсуждали в предыдущих главах.

Третьей основной проблемой является то, что процесс управления требованиями в значительной степени зависит от типа конкретной организации. В предыдущих главах мы обсуждали различные типы требований и взаимосвязи между ними. Однако сам процесс формирования и управления требованиями и их взаимосвязями будет отличаться в зависимсоти от организации.

На наш взгляд существует три основных типа организаций:

- **Организация-покупатель** - приобретает системы и использует их для своих собственных нужд. Главной задачей для организаций такого типа является разработка и управление пользовательскими требованиями, которые впоследствии используются для приемки системы.
- **Организация-поставщик** – отвечает на запросы организации-покупателя или организации-поставщика более высокого уровня. Организации такого типа обычно получают входящие требования и на их основе разрабатывают системные требования (и далее - спецификации системы).

  Следует заметить, что организация-поставщик может также выступать и в роли организации-покупателя, приобретая подсистемы или компоненты более низких уровней, но это уже немного другая форма приобретения, потому что в этом случае организация-поставщик обычно приобретает спецификации, а не систему.

- **Организация-производитель** – организация, которая разрабатывает и продает продукт. Организация такого типа формирует пользовательские требования для своих продуктов скорее под влиянием рынка, нежели общаясь с конкретными пользователями или организациями. Сбором требований в такой организации обычно занимается отдел маркетинга. Организация-производитель разрабатывает продукт в соответствии с пользовательскими (рыночными) требованиями и затем продает его.
  В сущности, организация такого типа одновременно выступает в роли и покупателя, и поставщика, однако взаимоотношения между разными департаментами, которые играют эти роли внутри данной организации, отличаются от стандартных взаимоотношений между обычной компанией-поставщиком и компанией-покупателем.

Мы еще вернемся позднее к этой теме в данной главе.

Четвертая проблема, которая делает процесс управления требованиями более трудным, чем управление другими задачами, заключается в сложности мониторинга процесса разработки требований. Один из трудных вопросов заключается в том, что необходимо понять является ли данный определенный набор требований полным, для того чтобы определить момент, когда следует остановиться. Не менее трудный вопрос состоит в том, чтобы правильно определить процент выполнения работ по формированию требований в те моменты времени, когда эта деятельность еще весьма далека от завершения.
Далее эта проблема еще более обостряется тем фактом, что необходимо оценивать качество разработанных требований. Возможно, к определенному моменту времени разработано достаточно много требований, но как руководитель может убедиться в том, что каждое из требований хорошо и правильно сформулировано? Как он может быть уверенным, что каждое из требований уникально, а все они вместе является необходимыми?

Заключительной проблемой является проблема постоянных изменений. Управление требованиями это в первую очередь управление изменениями. Обычно каждое предлагаемое изменение затрагивает одно или несколько требований. Однако влияние любого изменения зачастую очень трудно оценить, а без этого невозможно предсказать каким образом предложенное изменение повлияет на финальную стоимость продукта и сроки его разработки.

8.2.1 Выводы

Основные проблемы, возникающие при управлении процессом разработки требований, связаны с:

- планированием;
- контролем за ходом выполнения работ;
- контролем над изменениями.

Характер проблем, с которыми сталкиваются организации при управлении требованиями, зависит также и от специфики самой организации. Поэтому, дальнейшие разделы этой главы посвящены подробному описанию процессов разработки требований в организациях, типы которых упомянуты выше.
В заключительном разделе главы сформулированы некоторые общие подходы к управлению процессом разработки требований.

8.3 Управление требованиями в организации-покупателе

8.3.1 Планирование

Начальной точкой проекта в организации-покупателе является определение основной концепции. В самом минимальном варианте это может быть просто некая идея, хотя обычно концепция это нечто более подробно описанное и достаточно хорошо сформулированное. Для такой формализации есть очень простая причина – проект, прежде чем стартовать, должен быть утвержден; а для его утверждения необходим, следовательно, документ, доказывающий, что организация не зря потратит время и деньги (ресурсы). Такой документ обычно содержит краткое описание возможностей системы, т.е. то, что пользователи будут иметь возможность делать (концепция), и описание того, какие преимущества получит организация в результате реализации этого проекта.

Информация, которая содержится в документе, описывающим концепцию проекта, позволяет руководителю проекта начать планирование. Поскольку в концепции уже содержится описание того, что «*пользователи должны иметь возможность делать*», мы с самого начала получаем предварительный список заинтересованных сторон (*пользователей*) по отношению к системе, плюс один или несколько сценариев использования (*возможность делать что-то*).

Первым шагом при разработке плана является составление более полного списка типов заинтересованных сторон, а также более полного набора сценариев, покрывающих весь спектр операций, ожидаемых от системы, включая (если это необходимо) и режимы аварийного («неправильного») функционирования. После того, как определен финальный перечень всех типов заинтересованных сторон, можно переходить к детальному планированию того, каким образом будут формироваться требования.
Этот план может включать следующие действия:

1. Запланировать интервью с одним или несколькими представителями заинтересованных сторон каждого типа. Руководитель разработки требований должен убедиться в том, что руководители заинтересованных сторон утвердили участие своих подчиненных в процессе сбора требований. Очень важно, чтобы участие представителей заинтересованных сторон было именно утверждено (чтобы эти сотрудники смогли уделить для интервью свое рабочее время, а их руководство не было бы наказано за отсутствие в рабочее время подчиненных на своем рабочем месте). Руководитель разработки требований должен убедиться в том, что в интервью участвуют именно ключевые и наиболее опытные сотрудники. Зачастую руководители не желают отвлекать от текущих дел и отправлять на интервью самых компетентных сотрудников (самых полезных и наиболее информированных) из-за собственных краткосрочных интересов. В этом случае задачей руководителя разработки требований является убеждение руководителей заинтересованных сторон в большой потенциальной пользе выделения для интервью наиболее квалифицированных сотрудников.
2. Выделить время для записи как самих интервью, так и отчетов о них, а также согласовать эти документы с представителями заинтересованных сторон, т.е. с теми людьми, с которыми проводились интервью.

3. Определить стратегию интервью и согласовать ее с представителями заинтересованной стороны (по крайней мере с теми, кто, возможно, будет включен в процесс принятия решений). Стратегия интервью должна определять каким будет интервью, например, должен ли интервьюируемый сам формулировать пользовательские сценарии, или интервью будет посвящено обсуждению и анализу уже существующих сценариев, и т.д.
4. Перед интервью бывает полезно (но не всегда обязательно) собрать вместе всех кандидатов от заинтересованных сторон для того, чтобы объяснить цель предстоящих интервью. Такое общее собрание дает прекрасную возможность обсудить и разработать пользовательские сценарии, и лишний раз убедиться, что все типы заинтересованных сторон правильно определены.
5. Согласовать и задокументировать набор пользовательских сценариев, которые наилучшим образом описывают назначение и операции системы в ее окружении. Здесь очень важно убедиться в том, что пользовательские сценарии не являются слишком ограниченными по содержанию и диапазону области действия.
6. После проведения интервью, необходимо на основании отчетов сформулировать пользовательские требования и согласовать их с участниками интервью.
7. Продумать, создать и согласовать структуру пользовательских требований, в которую они будут в дальнейшем размещаться.
8. Разместить каждое из полученных пользовательских требования в разработанную структуру (при необходимости структура может изменяться).
9. Определить и зафиксировать любые ограничения. Некоторые ограничения могут формироваться сразу на уровне требований, например, геометрические размеры будущего продукта, и тогда эти ограничения (относящиеся к продукту) должны быть с самого начала зафиксированы в пользовательских требованиях. Другими ограничениями могут быть, так называемые, плановые ограничения, т.е. ограничения, относящиеся к самому проекту (например, бюджетная стоимость проекта, сроки его реализации, ресурсы, качество) и тогда они должны быть зафиксированы в проектном плане и в дальнейшем влиять на планирование.
10. Решить, нужно ли вводить дополнительные атрибуты, чтобы поддержать содержание требований. Многие организации практикуют обязательное применение некоего стандартного набора атрибутов для требований; в других же - применение набора атрибутов носит просто рекомендательный характер. В качестве примера стандартных атрибутов можно перечислить следующие: приоритетность, срочность, статус, метод проверки и критерии приемки.
11. Согласовать критерии рецензирования и анализа требований, как каждого индивидуально, так и некоего набора в целом. Эти критерии лучше всего свести в один перечень, который будет в дальнейшем использоваться аналитиками. В идеальном варианте, - чем раньше вы создадите такой перечень и разошлете его всем аналитикам, разрабатывающим требования, тем лучше. Это позволит им в самом начале представить, что именно от них ожидается и, соответственно в дальнейшем, сократить время на согласование требований.
12. Определить процесс рецензирования требования и увязать его со значением статуса последнего. Пример процесса рецензирования представлен на рис. 8.2 в виде диаграммы переходов состояний статуса требования. Начальным значением статуса требования является значение «Предложено». Требование в этом статусе должно быть

рецензировано командой разработчиков требований и только после положительного результата оно может перейти в статус «Проверено». После этого шага требование проверяется командой заказчика, после чего может перейти в статус «Одобрено».

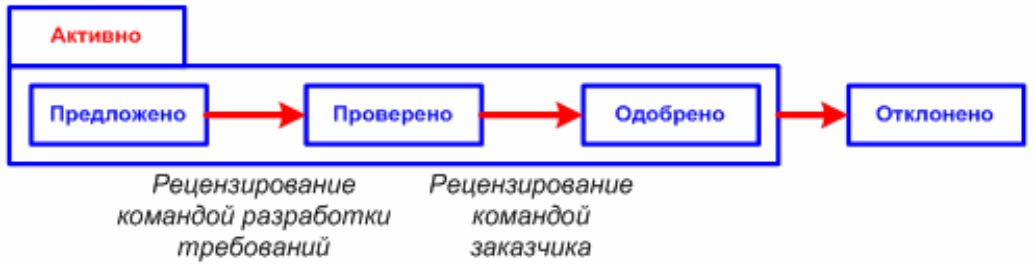


Рис. 8.2 Диаграмма переходов состояний статуса пользовательского требования.

Необходимо отметить, что в любой момент требование, находящееся в фазе «Активно», может сменить свой статус на «Отклонено».

Важно, чтобы критерии проверки требования на каждом шаге возможной смены значения его статуса, были бы обязательно определены заранее.

13. Выполнить рецензирование в соответствии с разработанным процессом и критериями рецензирования.

Как видно, приведенный выше список задач предполагает необходимость принятия целого ряда организационных решений. Формирование и принятие этих решений входит в обязанности руководителя процесса разработки требований, поскольку он должен согласовать этот план с представителями заинтересованных сторон (участниками интервью), их руководителями, основным заказчиком проекта.

Все ограничения, относящиеся к планированию, необходимо оценить с точки зрения их целесообразности и осуществимости. Например, заинтересованные стороны, как всегда, будут настаивать на вводе системы в эксплуатацию в очень сжатые сроки и с низкой себестоимостью, хотя в реальности такое может быть просто невозможно; что и требуется оценить.

Одним из ярких примеров нереальных ограничений по срокам выполнения является проект по разработке системы управления службой скорой помощи Лондона, выполнявшийся в начале 90-х. Руководителям службы эта система нужна была для того, чтобы предоставлять властям необходимую статистику работы организации. В результате, в качестве ограничений были установлены очень короткие сроки реализации проекта и назначена ранняя дата вввода системы в промышленную эксплуатацию. Соблюсти такие сроки было невозможно. Многие компании-разработчики, участвовавшие в тендере, пытались уговорить руководство службы скорой помощи увеличить сроки реализации проекта и отодвинуть дату вввода системы в эксплуатацию, но руководство службы скорой помощи было непреклонно. В результате большинство компаний, в конце концов, отказалось от участия в тендере, а оставшиеся (менее опытные разработчики) все еще пытались выдержать поставленные ограничения по срокам. В итоге проект был полностью провален. И более того, этот проект принес ущерб многим людям.

Этот пример лишний раз подтверждает, что реализм в планировании является признаком честности и профессионализма руководителя.

8.3.2 Контроль за ходом выполнения работ

Контроль за ходом выполнения работ может начинаться сразу же после утверждения плана. Очевидными точками контроля являются даты завершения промежуточных этапов плана. На более ранних стадиях работ выполнение промежуточных задач связано, в основном, с подготовкой и проведением интервью, а также документированием их результатов. Эти задачи достаточно просто контролировать в процессе их выполнения.

Для успешного мониторинга за всем остальным процессом могут помочь три основные контрольные точки:

- определение структуры спецификации требований;
- определение атрибутов каждого из требований;
- определение процесса рецензирования требований в соответствии с перечнем критериев рецензирования.

Если структура спецификации требований сформирована, то становится достаточно легко определить, где требования уже сформулированы, а где в структуре (в документе) пока еще остаются незаполненные места. Такие «дыры» можно легко найти и принять меры к их устранению.

Если набор атрибутов для требований определен, то уже намного легче контролировать процесс заполнения значений этих атрибутов для каждого требования.

В процессе рецензирования тоже достаточно просто осуществлять мониторинг, используя для этого результаты проверки требований на соответствие критериям рецензирования, а также их текущий статус.

8.3.3 Изменения

В процессе разработки пользовательских требований обязательно присутствует период быстрых и интенсивных изменений. Очевидно, что на этом этапе не имеет большого смысла вводить формальную процедуру управления изменениями, поскольку ситуация на этом этапе весьма динамична и изменения могут вноситься, что называется, «на ходу». Однако задача руководителя разработки требований состоит еще и в том, чтобы вовремя определить момент, когда требования становятся более стабильными, а значит пора вводить формальную процедуру управления изменениями требований.
Обычно этот момент наступает тогда, когда все требования прошли полный цикл рецензирования и перешли в статус «Одобрено» (см. рис. 8.2).

Управление изменениями является жизненно необходимой дисциплиной в разработке требований. Формальность использования процесса внесения изменений зависит от стадии, на которой в данный момент находиться проект. Здесь можно выделить следующие стадии:

- использование пользовательских требования для конкурентного выбора поставщика;
- заключение контракт на разработку системы;
- завершение разработок спецификаций и начало производства системы;
- выполнение приемочных тестов и эксплуатационных испытаний системы;
- эксплуатация системы в промышленных условиях.

Данный список определяет последовательность контрольных точек, на каждой из которых степень ответственности за ранее принятые решения постепенно увеличивается. Следовательно, чем дальше продвигается проект, тем более формальной должна становиться процедура внесения изменений, и тем выше становятся затраты, связанные с реализацией предлагаемых изменений.

На какой бы стадии не находился ваш проект, процесс управления изменениями требует обязательного выполнения следующих шагов:

1. зафиксировать предлагаемое изменение;
2. определить влияние предложенного изменения;
3. решить - принять или отклонить предложенное изменение - и, если принять, то:
4. определить момент, когда нужно реализовывать предложенное изменение.

Предлагаемое изменение должно содержать обоснование изменения (причину) и ссылку на требование (возможно, на несколько), которое должно быть изменено, удалено или добавлено. Необходимо, чтобы и имя автора, вносящего изменение, было зафиксировано как один из обязательных параметров.

На втором шаге, влияние, потенциально оказываемое предлагаемым изменением, будет зависеть в первую очередь от этапа, на котором в данный момент находиться проект. Для оценки влияния предлагаемого изменения необходима информация о том, каким образом и насколько требование, изменения к которому предложены, затрагивает идущие «вниз» информационные потоки, т.е. как это требование связано с системными требованиями, функциональными требованиями или спецификациями, а может уже и с разработанной частью системы, или функциональностью системы, находящейся уже в эксплуатации.

На третьем шаге решение должен уже принимать специальный комитет (группа) по внесению изменений. Состав и форма организации работы такого комитета зависят от специфики компании, размера проекта, стадии разработки системы или стадии ее промышленной эксплуатации.

Если изменения согласованы и приняты, то процесс переходит на четвертый шаг. В некоторых случаях требуется, чтобы изменение было внесено и выполнено незамедлительно, независимо от стоимости. В других случаях изменение может быть отложено до реализации следующей версии системы. Несомненно, что процесс внесения изменений может содержать любое количество промежуточных этапов, наличие которых диктуется спецификой производствai и сложившимися обстоятельствами.

Всегда полезно представлять процесс управления изменением в виде диаграммы состояний (диаграммы переходов состояний). Рис. 8.3 демонстрирует такой пример.

Также важно с самого начала определить, будет ли меняться значение статуса требования, в отношении которого вносится изменение, чтобы отобразить этот факт. Существует два разных подхода к решению данного вопроса. Одна группа специалистов считает, что взаимосвязь между изменением и самим требованием, так или иначе, храниться в рамках предложения на внесение изменения и, следовательно, нет нужды менять статус самого требования. Другая группа считает, что если принимается решение внести изменение, то конкретное требование как раз и становится тем объектом, в который вносится изменение, а, следовательно, значение статуса требования также должно меняться, чтобы по статусу самого требования можно было бы в данный момент

определить значение статуса предложенного изменения. (Именно такой подход применялся нами в главе 2.)
Но какой бы подход вы не использовали, необходимо заранее определить набор значений, используемых для индикации статуса изменений, и предварительно решить, будут ли эти значения каким-то образом влиять на статус рецензирования затрагиваемого требования (будут ли они изменять его статус).

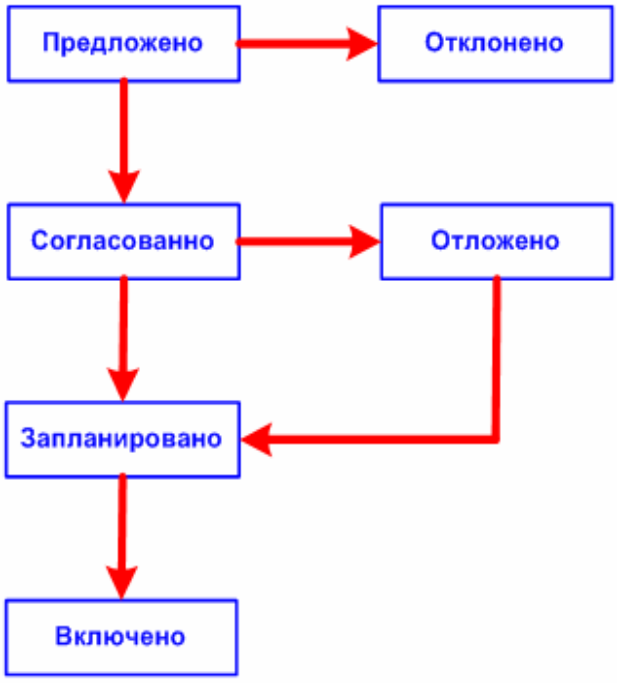


Рис. 8.3 Диаграмма переходов состояний управления изменениями.

В заключение перечислим основные отличительные черты процесса управления требованиями в организации–покупателе.

Работа с требованиями сосредоточена, в основном, в области разработки пользовательских требований. Этот творческий процесс и поэтому в самом начале весьма трудно точно определить его ресурсные и временные границы. Однако после определения всех заинтересованных сторон, разработки и согласования всех пользовательских сценариев, планирование может производиться более аккуратно.

В самом начале работы процесс внесения изменений имеет незначительную степень формализации, которая постоянно увеличивается по мере развития проекта.

8.4 Управление требованиями в организации-поставщике

Организации-поставщики работают по запросам заказчиков, разрабатывая системы и их компоненты. Для того, чтобы получить контракт на выполнение этой работы организация-поставщик должна подготовить коммерческое предложение, содержащее описание тех подходов и методологий, которые она планирует использовать для выполнения работ, а

также предварительную оценку срока реализации проекта и его стоимости. Зачастую заказчик проводит тендер, и тогда запрашиваются предложения сразу от нескольких организаций-поставщиков, которые соревнуются за право выиграть тендер. Следовательно, деятельность такой компании-поставщика полезно рассмотреть с двух разных точек зрения – борьба за победу в тендере (подготовка коммерческого предложения) и выполнение проекта в случае победы.

8.4.1 Подготовка коммерческого предложения

Данный раздел описывает аспекты управления процессом создания коммерческого предложения, как ответа на список требований заказчика.

Планирование

Очень часто начальной точкой работ по проекту внутри организации-поставщика является приглашение к участию в тендере (invitation to tender = ITT) или запрос на коммерческое предложение (request for proposal = RFP). Такое приглашение или запрос содержат целый список требований, которым должна удовлетворять будущая система.

Характер полученных требований будет зависеть от типа клиентской организации, которая прислала приглашение к участию в тендере. Если клиентская организация является конечным потребителем, заказывающим систему для собственных нужд, то, вероятнее всего, полученные требования будут являться пользовательскими. В другом случае клиентская организация может быть сама поставщиком (разработчиком), желающим найти субподрядчика для выполнения части работ, например, для разработки подсистем и\или компонентов системы. Тогда, вероятнее всего, организация-поставщик получит на входе системные требования и определенные ограничения относительно их реализации. Для того, чтобы избежать путаницы и сделать дальнейшее изложение более понятным и наглядным, мы будем называть требования, полученные от клиентской организации, *входящими*, независимо от их реального характера.

Какой бы ни была суть полученных входящих требований, в первую очередь, мы должны проанализировать их, чтобы определить являются ли требования:

- четко выраженными и отделенными от информации описательного характера;
- однозначными (недвусмысленными);
- непротиворечивыми (совместимыми);
- свободными от чрезмерных ограничений, серьезно препятствующих успешной реализации проекта.

Одним словом, необходимо убедиться, что предоставленные требования могут служить достаточно прочной основой для разработки коммерческого предложения.

С точки зрения планирования, самым первым показателем предстоящего объема работ является количество полученных требований.

В процессе оценки и анализа входящих требований необходимо постараться выявить возможные проблемы и предложить потенциальные способы их решения, например, измененить формулировки некоторых требований или даже предложить взамен другие

требования, которые, вполне возможно, могли бы быть удовлетворены уже готовым («коробочным») компонентом.

После завершения оценки и анализа требований, все обнаруженные в требованиях проблемы должны быть направлены заказчику. Далее возможно проведение переговоров с заказчиком с целью согласования предлагаемых изменений. Наличие такой возможности зависит от условий тендера. Если предложение к участию в тендере направлено только одному поставщику, то это существенно облегчает процесс согласования входящих требований, который может начинаться сразу же. В том случае, если в тендере участвуют несколько поставщиков, компании-участнику необходимо вести себя более осторожно. Это связано с тем, что все вопросы и ответы на них компания-организатор обычно рассылает сразу всем участникам тендера. Следовательно, задавая вопросы, можно подбросить конкурентам ценные идеи и информацию, даже не желая этого. Поэтому в подобной ситуации необходимо все найденные проблемы обсудить вначале внутри компании-поставщика и затем уже решать, стоит ли выносить их на общее обозрение.
В этой ситуации можно рекомендовать три основных варианта развития событий:

- полностью проигнорировать проблему;
- сделать какое-то предположение (допущение), позволяющее устранить проблему, и, зафиксировав его документально, двигаться дальше;
- принять решение, что, независимо от последствий, необходимо запросить заказчика.

В последнем случае нужно попытаться сформулировать запрос таким образом, чтобы дать конкурентам как можно меньше ценной информации.

Параллельно с оценкой входящих требований необходимо начинать работу над предлагаемым решением. Очевидно, что конечным результатом этой работы явится готовое для отправки заказчику коммерческое предложение. Существует много разных подходов к разработке коммерческого предложения, но очевидно, что, в любом случае, коммерческое предложение должно содержать четкое описание того, каким образом будут удовлетворяться все требования заказчика.

Руководитель процесса подготовки коммерческого предложения (тендер-менеджер) должен назначить ответственных для подготовки предложений по каждому из требований заказчика. В результате такой работы очень важно получить целостное и согласованное предложение, а не случайный набор слабо связанных между собой частей. Наилучший способ получить хорошее предложение это создать общую модель, которая будет являться основой для предлагаемого решения.
В зависимости от характера предложение модель может быть абстрактной и являться основой для разработки системных требований, или модель может быть более детализированоой и определять основную концепцию архитектуры системы. Каждый из специалистов, отвечающих за подготовку своей части предложения, будет наполнять модель необходимыми деталями. Такой подход дает возможность устанавливать связи между входящими требованиями и моделью (ее частями), что, в конечном итоге, обеспечивает целостность предложения и устраняет несогласованности в нем. Однако этот подход не может гарантировать отсутствие других проблем – ведь даже в этом случае специалисты, готовящие предложение, вынуждены работать с неполной информацией, базируясь на определенных допущениях и постоянно гадая на тему, - «что же на самом деле имел в виду заказчик». Однако никуда не денешься, - такова жизнь!

По окончании работ очень важно, чтобы специалисты, участвовавшие в подготовке тендерного предложения, зафиксировали всю информацию, собранную в ходе подготовки предложения.

Зачастую команда, занимающаяся подготовкой предложения, находиться под большим прессингом, поскольку им необходимо подготовить и отослать заказчику предложение к определенной дате. Естественно, что после отсылки предложения люди стараются взять передышку и часто забывают записать всю информацию, собранную в процессе подготовки, хотя эта информация представляет большую ценность для команды, которая, возможно в последствии, будет выполнять этот проект.

Для больших проектов объем информации, которую собирает команда, может быть просто огромным, как и интервал времени между отсылкой предложения и началом реализации системы после победы в тендере (например, 6–8 месяцев). В подобном случае, еще более важно фиксировать всю собранную информацию в процессе подготовки предложения, поскольку специалисты, принимавшие участие в разработке предложения, могут и не войти в состав команды, которая будет реализовывать проект. Или - в случае включения в команду разработки - эти специалисты по истечении столь большого промежутка времени могут просто забыть часть информации или некоторые ключевые допущения и идеи.

Следующим важным этапом является заключение договоров с поставщиками (субподрядчиками). Обычно такие договора заключаются на условии вступления их в силу в случае победы в тендере. Привлечение субподрядчиков для участия в подготовке предложения оказывает влияние на степень детализации готовящегося решения. Основой для соглашений между подрядчиком и субподрядчиком является набор требований для компонента или подсистемы; при этом организациям необходимо договориться о степени детализации требований. Обычно степень детализации требований зависит от предыдущего опыта совместной работы организаций, их доверия их друг другу. (См. процесс согласования, приведенный в главе 2.)

Контроль за ходом выполнения работ

Общий контроль и оценка степени (процента) выполнения работ при подготовке коммерческого предложения являются важным фактором, поскольку дата приема тендерных предложений жестко зафиксирована и ее невозможно сдвинуть на более поздний срок. Заключительной точкой подготовки предложения можно считать момент, когда последнее достаточно точно поясняет, каким образом удовлетворяется каждое из требований заказчика. Однако одних только утверждений и пояснений, как удовлетворяются требования, явно не достаточно. Теперь необходимо убедиться, что все эти утверждения правомерны и обоснованы. Это уже, в большей степени, относится к анализу процесса, но именно это дает возможность оценить степень выполнения работ с помощью такого показателя, как, например, процент входящих требований, которые взаимосвязаны с моделью предлагаемого решения (и, следовательно, или с системными требованиями или спецификациями дизайна).

Другим параметром, который можно использовать для оценки работ, является объем невыполненной работы. В целом общий объем оставшейся и предстоящей работы можно определить, например, по числу входящих требований, с которыми связаны пока еще нерешенные проблемы, или\и по числу входящих требований, которые еще ждут своего решения.

Другой важной контрольной точкой хода работ является создание общей модели решения, с которой будет работать вся команда. Вот почему одной из важных задач руководителя работ является быстрое создание и согласование общей модели решения.

В дополнении к всем вышеупомянутым способам мониторинга, необходимо контролировать еще и качество системных требований. Это может производиться аналогично тому, как описано выше для случая организации-покупателя (часть 8.3), которая отслеживает создание пользовательских требований. Т.е. точно так же необходимо разработать критерии проверки требований, точно также для контроля хода проверки необходимо определить соответствующие значения статуса требования, и выполнять анализ и рецензирование в соответствии с этими критериями и значением статуса.

Изменения

Можно отметить следующие источники изменений, характерные для процесса подготовки коммерческого предложения:

- заказчик;
- поставщики (субподрядчики);
- внутренние источники.

Можно, конечно, предположить, что заказчик не будет менять условия тендера (требования) в процессе подготовки коммерческого предложения, и в идеале это предположение справедливо. Однако, для собственной же пользы, на это не стоит надеяться. Обычно вероятность возникновения изменений связана прямопропорциональной зависимостью с размером создаваемой системы (или компонента). Для очень больших систем поставщики обычно начинают подготовку своих коммерческих предложений сразу же после получения первой черновой версии запроса на коммерческое предложение (RFP). Это необходимо для того, чтобы раньше включиться в процесс подготовки и заставить команду двигаться в нужном направлении. Разумеется, что с течением времени заказчик создает новые версии требований, которые могут содержать изменения и которые приходится отрабатывать, внося соответствующие коррективы в работу.

Самой первой задачей после получения новой версии RFP (или просто отдельного списка требований) является определение объема и характера изменений. В зависимости от желания заказчика и способа рассылки новой версии запроса (или списка требований) изменения могут быть как-то отмечены в документе, а могут быть совершенно «невидимы». Тем не менее, каждое из обнаруженных изменений должно быть соотнесено с уже выполненной работой, после чего должен быть оценен объем и план новых работ, и лишь после этого можно приступать к переделкам, если это необходимо.

Изменения от заказчика могут поступать и косвенным образом, - как ответы на вопросы, заданные любым участником тендера. В этом случае изменения достаточно хорошо выражены и их легко оценить.

Изменения, поступающие от поставщиков (субподрядчиков), более вероятны. Зачастую изменения поступают сразу и из-за того, что субподрядчик находит во входящих требованиях такие, которые он никак не может удовлетворить. Или изменения могут поступить позднее, если субподрядчик обнаружит, что он не может выполнить то, что вначале не вызывало у него сомнений.

Ситуация с изменениями, поступающими от внутренних источников, мало чем отличается от вышеописанного – внутренние изменения возникают по тем же самым причинам. Первоначальные предположения и допущения могут в какой-то момент оказаться несостоятельными и, следовательно, необходимо будет искать альтернативные пути и решения. Но чтобы ни являлось причиной и источником изменений, необходимо всегда стараться держать обновленными:

- входящие требования;
- требования для поставщиков и субподрядчиков (можно назвать их исходящими);
- допущения, предположения и интерпретации, сделанные командой, разрабатывающей коммерческое предложение.

8.4.2 Выполнение проекта

Планирование

Выполнение работ по проекту начинается тогда, когда уже имеется согласованный контракт, который основывается на коммерческом предложении, ранее принятом заказчиком и согласованным с ним на этапе переговоров и заключения контракта. Вполне возможно, что в процессе работы была собрана ценная информация, не вошедшая в коммерческое предложение. Например, это могут быть детально прописанные требования, допущения, предварительная или детальная архитектура решения, начальные оценки рисков, связанных с разработкой системы. Все это вместе взятое может использоваться для оценки трудозатрат и сроков реализации проекта.

На стадии разработки все задачи должны быть определены более подробно, нежели чем на стадии подготовки коммерческого предложения. Более того, само коммерческое предложение, принятое заказчиком, на стадии разработки может уже рассматриваться как часть входящих требований.

Совокупность информации, которая рождается на стадии разработки, разумеется, зависит от характера проекта и специфики организации, однако в любом случае она должна включать в себя модель решения. Модель может разрабатываться в два этапа: вначале может быть создана абстрактная модель, а в последствии уже более подробная модель потенциального решения. В случае, если существует возможность разработки нескольких альтернативных решений, то необходимо разработать критерии для оценки этих решений с целью определения по какому единственному пути двигаться дальше. В процессе обсуждения и оценки различных вариантов решений неизбежно появление разных мнений и противоречий, устранить которые можно лишь достижением компромисса. Процесс достижения компромиссного решения может проходить либо целиком внутри организации-поставщика, либо с привлечением заказчика и/или субподрядчиков.

На стадии разработки необходимо постоянно контролировать, чтобы все системные требования соответствовали входящим контрактным требованиям и далее - адекватно удовлетворялись предлагаемым решением. Необходимо также отслеживать, чтобы при переходе от одного уровня требований к другому никакая информация не терялась. При этом уровень детализации будет последовательно увеличиваться, чтобы на самом детализированном уровне уже нечего было расписывать.

Весьма важно быть уверенным в том, что средства тестирования каждого требования (другими словами, средства и возможности продемонстрировать то, что требование удовлетворено) однозначно понятны и документально зафиксированы.

Внимательный анализ доступной информации с целью определения ее объема и качества должен являться первым шагом стадии разработки. В идеальном варианте вся эта информация уже собрана командой, разработавшей коммерческое предложение, и подготовлена для использования командой разработки. Однако в реальной жизни этого не происходит и зачастую важная информация теряется, что приводит в последствии к большому разрыву между намерениями команды подготовки коммерческого предложения и тем, что в итоге делает команда разработки. А это, в свою очередь, ставит организацию в рискованное положение.

На следующем шаге руководитель проекта должен обязательно выявить различия между коммерческим предложением, ранее принятым заказчиком, и заключенным контрактом. Затем необходимо определить какое влияние эти обнаруженные изменения могут оказать на разработанную документацию и, возможно, пересмотреть план работ, внести соответствующие изменения в системные требования, спецификации системы и спецификации отдельных компонентов системы.

Не мешает также обсудить с заказчиком оставшиеся допущения и замечания (хотя лучше было бы это делать в процессе обсуждения контракта).

Следующим важным вопросом, возникающим при планировании работ, является выбор варианта поставки системы. Либо вся система будет реализована за один этап, либо предусматривается серия релизов (с постепенным наращиванием функциональных возможностей до максимума к финальному релизу). Разбиение реализации системы на этапы позволяет заказчику получить первичный функционал системы гораздо раньше. Этот подход пользуется большой популярностью в сфере разработки программного обеспечения, где в начале разработки всегда возникают сомнения в полезности или необходимости тех или иных функций приложения (что может быть легко скорректировано в последующих релизах).

С точки зрения управления требованиями каждый релиз системы должен строиться на некотором наборе требований, которые и предполагается воплотить именно в данном релизе. Такой подход может быть легко реализован, если у каждого требования появится специальный «релизный» атрибут. Это может быть атрибут либо булева типа, либо перечислимого. Для перечислимого типа возможный список значений может выглядеть следующим образом:

{ TBD, Релиз 1, Релиз 2, Релиз 3 } ,

где TBD (*to be decided* = «необходимо определить») обычно является значением по умолчанию, т.е. определяет некий первичный набор реализованных требований. Для булева типа атрибута, как известно, возможно иметь всего два значения, поэтому атрибут такого типа применяют, в случае если запланировано всего два релиза системы.

Контроль за ходом выполнения работ

Контроль за ходом реализации требований в процессе разработки должен быть направлен, в первую очередь, на оценку объема и качества производимой информации. Очень важно контролировать сколько времени и усилий было затрачено и сколько еще предполагается

затратить для завершения работы. Это позволяет постоянно сверяться с разработанным планом и давать прогнозы относительно того, будет ли работа завершена своевременно. Конечно, чтобы давать такие прогнозы, кроме фактических данных требуется еще и опыт руководителя.

Если становится понятно, что работа начинает отставать от плана, руководителю необходимо принять адекватные корректирующие действия. В этом случае возможно либо изменение сроков выполнения промежуточных задач, либо перераспределение ресурсов между задачами (например, включение дополнительных специалистов в команду).

Еще одной задачей руководителя является контроль за актуальность (обновлением) всей проектной документации. Это особенно важно при внесении изменений во входящие требования. Необходимо, чтобы производные требования и, соответственно, все связи между входящими и производными требованиями были также скорректированы в соответствии с принятыми изменениями. Это же относится и к обновлению требований и связей между ними на всех других уровнях.

Изменения

Причины возникновения изменений на стадии разработки ничем не отличаются от трех перечисленных ранее причин возникновения изменений на этапе подготовки коммерческого предложения. Хотя объем изменений со стороны заказчика здесь обычно меньше, чем на этапе тендера. А вот объем внутренних изменений и изменений со стороны субподрядчиков, напротив, гораздо больше. Процедура определения характера и последствий изменений также ничем не отличается от описанной ранее. Однако следует заметить, что изменения, вносимые в процессе разработки, могут иметь гораздо более серьезные последствия. Небольшие изменения в процессе тендера могут быть легко приняты всеми сторонами без каких-либо дополнительных затрат. Более серьезные изменения могут привести лишь к корректировке условий договора. В то время как изменения, вносимые в процессе разработки, обычно приводят к изменению сроков и стоимости проекта. После определения возможных последствий изменений, задачей руководителя является принятие решения о том, может ли компания взять на себя дополнительные затраты по реализации этих изменений или же необходимо обсудить их с заказчиком и\или субподрядчиками.

Если же изменение принимается, а при этом проект предусматривает поставку системы в несколько релизов, то дополнительной задачей для руководителя является принятие и согласование решения о том, в каком из релизов будет воплощаться предложенное изменение.

Суммируя все вышесказанное, можно отметить следующие основные положения. Организации-поставщики, участвуя в тендере, готовят коммерческие предложения и, если выигрывают тендер, приступают к разработке системы. Хорошо разработанные входящие требования невероятно важны как основа для дальнейшей разработки системы. Для успешного выполнения проекта необходимо всегда иметь «свежие» (обновленные) версии входящих требования. Наличие связей (трассировки) между входящими требованиями, производными требованиями всех уровней, требованиями для субподрядчиков, тестовой информацией позволяет правильно оценивать влияние предлагаемых изменений, и всегда иметь представление о ходе выполнения работ по проекту.

8.5 Управление требованиями в организации-производителе

Организации-производители сами разрабатывают пользовательские требования и создают систему, которая им соответствует. Следовательно, организации этого типа имеют много общего и с организациями-покупателями, и организациями-поставщиками. Основное отличие заключается лишь в том, что отношения «заказчик-поставщик» остаются в рамках одной организации и эти роли выполняют разные департаменты компании.

8.5.1 Планирование

Продукт с одной версией

Планирование разработки одного продукта, и к тому же имеющему всего одну версию, аналогично планированию в компании-покупателе и компании-поставщике. Однако, на этапе формирования предложения или на этапе разработки уже может появляться заметная разница. Например, прежде, чем начать разрабатывать новый продукт, компания пожелает иметь начальное представление о том, во что ей это может обойтись. Для чего ей самой потребуется сформировать первичные пользовательские требования и разработать концепцию решения.

Разработка пользовательских требований происходит также, как и в организации-покупателе. Аналогичным образом необходимо определить список заинтересованных сторон и разработать пользовательские сценарии. Однако, вместо того, чтобы проводить интервью с реальными пользователями и представителями других заинтересованных сторон, сотрудники компании зачастую сами играют эти роли. Т.е. они «ставят себя на место» представителей заинтересованных сторон и пытаются с этой позиции сформулировать пользовательские требования. С точки зрения планирования такой подход не имеет больших отличий: заинтересованные стороны должны быть определены, интервью должны быть проведены, требования должны быть собраны, структурированы и проверены на соответствие с разработанным критериям.

Разработка первоначального чернового варианта решения во многом схожа с процессом подготовки коммерческого предложения. Основное отличие заключается лишь в большей доступности людей, которые разрабатывали пользовательские требования. Это дает возможность тесно взаимодействовать с ними на этапе разработки, намного облегчая процесс внесения изменений в пользовательские требования, что позволяет упростить разработку, а, следовательно, ускорить время выхода продукта на рынок и уменьшить затраты на его разработку. В случае обнаружения неоднозначностей или противоречий в требованиях они уточняются гораздо быстрей и проще. Это иногда даже позволяет (не выходя за рамки первоначального бюджета) улучшить общие характеристки системы, поскольку разработчики могут обсуждать с «пользователями» даже технические решения. Возможно, весь этот процесс покажется вам весьма неформальным, но зачастую это можно допустить. Хотя определенная степень формальности все-таки должна быть оговорена и утверждена до начала работ.

После согласования пользовательских требований и выработки чернового варианта решения, организация-производитель может принять решения о прекращении работ или, наоборот, решить инвестировать дополнительные средства и продолжить разработку более детальных спецификаций или даже прототипа системы. Таким образом, процесс разработки будет проходить в несколько стадий и каждая последующая стадия будет базироваться на

результатах предыдущей. Соответственно, у каждой стадии будет свой бюджет и свои собственные цели, а по завершению каждой из них должна производиться проверка на соответствие полученных результатов поставленным целям и бюджету. Данная последовательность действий может быть продемонстрирована с помощью концепции «ворота стадий», изображенной на рис. 8.4.

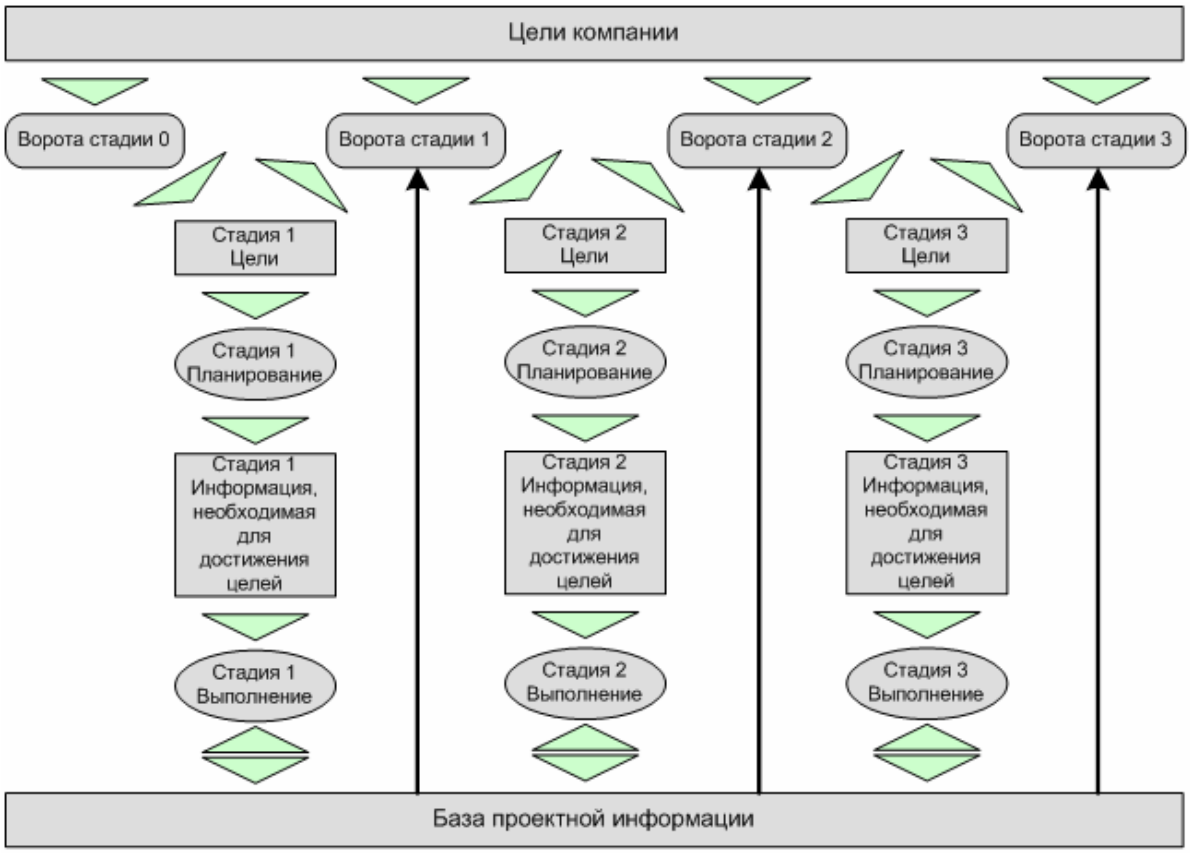


Рис. 8.4 «Ворота стадий» и работа над проектом.

На входе в первые ворота (Ворота стадии 0) определяются цели, а также бюджет и сроки проекта. На следующем шаге эта информация подпитывает процесс планирования, в рамках которого определяется характер и суть информации, которую необходимо сгенерить, чтобы достичь целей данной стадии; здесь же определяется и план работ, необходимых для достижения поставленных целей в срок и в рамках отведенного бюджета. На этом шаге целью может быть только лишь первичное исследование черновой концепции или предварительная оценка рынка сбыта, и т.п. А вот в конце этой стадии компания должна проанализировать полученные результаты на соответствие поставленным в начале целям с тем, чтобы определить стоит ли продолжать проект или его следует остановить. В процессе этой оценки необходимо принимать во внимание тот факт, что бизнес-задачи (цели) компании могли измениться в период начальной стадии.

Если принято решение продолжить проект, то следующая стадия (Ворота стадии 1) может начаться только после того, как для нее определены собственные цели, бюджет и

временные рамки. Для второй стадии может быть, например, принято решение сделать коммерческое предложение с подробным расчетом стоимости и более детальным исследованием объемов рынка сбыта.

Можно предположить, что целью следующих «ворот» будет оценка предполагаемой прибыли от реализации системы по отношению к предполагаемым затратам. Это опять же приведет к необходимости принятия решения относительно целесообразности продолжения работ по проекту. Если будет решено продолжить инвестировать проект, то возникнет необходимость принять решение об объеме работ для следующей стадии, например, это может выглядеть так:

- более подробно оценить стоимость разработки и производства системы;
- разработать прототип;
- создать упрощенные комплекты системы и раздать их на пробную эксплуатацию реальным потребителям (α-, β-тестирование);
- запустить производство в полном объеме;
- и т.д.

Таким образом, концепция «ворота стадий» позволяет организации в любой момент времени выполнять работы только в рамках одной стадии с высокой степенью ответственности за соблюдение договоренностей относительно выделенного бюджета и ресурсов. Такой подход позволяет организации контролировать свою инвестиционную стратегию и размеры инвестиций, управляя показателем возврата инвестиции.

Несколько продуктов и версий

Для организации-производителя вполне типична ситуация, когда она разрабатывает одновременно несколько версий одного и того же продукта и в каждый момент времени все эти версии находятся на разных стадиях разр. Обычно, в таком случае, часть версий продукта находится в эксплуатации у потребителей, другая часть находиться в процессе разработки, а третья часть находиться лишь на этапе планирования. С точки зрения управления и планирования каждая версия может рассматриваться как отдельный проект со своими собственными стадиями и «воротами». Однако при этом необходимо, чтобы планирование учитывало все версии продукта, находящиеся на разных стадиях жизненного цикла. Необходимо заранее запланировать, в какой момент текущая версия продукта, находящаяся в коммерческой эксплуатации, будет заменена новой версией. При использовании метода «ворота стадий» все эти аспекты необходимо учитывать в процессе формировании целей стадий, чтобы находить наилучшую инвестиционную стратегию, направленную на увеличение рыночной доли компании.

Еще один фактором, который необходимо принимать во внимание, является возможное наличие различных версий продукта, предназначенных для различных рынков. Например, необходимо иметь продукт, пользовательский интерфейс которого поддерживает разные языки, чтобы иметь возможность продаваться в разных странах мира.

Для того, чтобы избежать путаницы, мы введем понятие «модификации», которое будет означать «*нечто, отличающееся по форме или в деталях от чего-то базового (главного)*». Следовательно, у нас может быть базовый продукт (лучше назвать это «ядром продукта»), который имеет пользовательский интерфейс на английском языке, а также его модификации с пользовательскими интерфейсами на французском, немецком и испанском

языках для региональных рыков сбыта. При этом каждая модификация может иметь свои собственные версии. Т.е. каждая следующая версия (в рамках одной модификации) будет чем-то положительно отличаться от предыдущей (исправлены ошибки и/или добавлены новые возможности).

На рис. 8.5 схематично показаны несколько параллельных модификаций одного и того же продукта, каждая их которых имеет несколько версий, находящихся на разных стадиях своего жизненного цикла. Буквы S, D, U обозначают различные фазы разработки продукта – разработку спецификаций (specification), производство (development) и коммерческую эксплуатацию (use). С точки зрения метода «ворота стадий», каждой из этих фаз соответствует одна или несколько стадий жизненного цикла продукта.

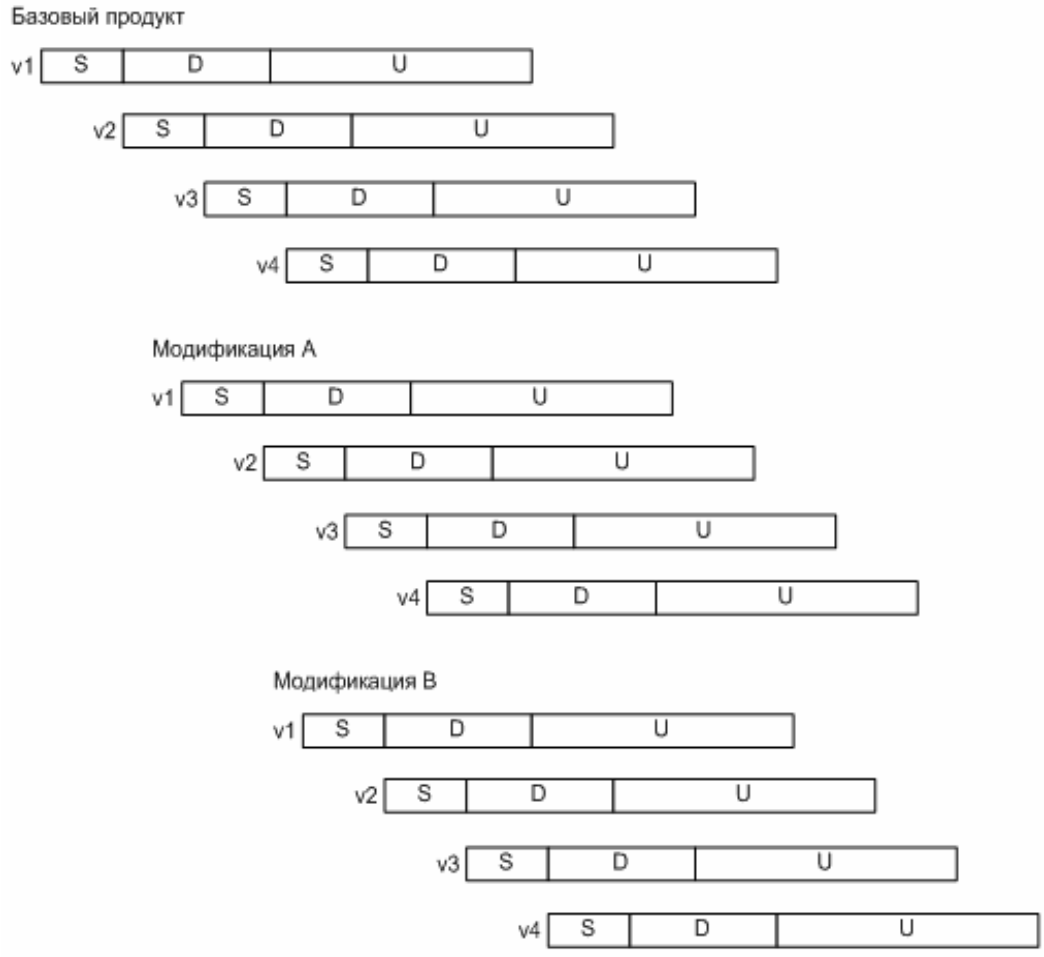


Рис. 8.5 Версии и модификации.

С точки зрения управления требованиями это означает, что каждая модификация будет иметь много общих требований с базовым продуктом, но наряду с некоторым количеством требований, которые будут присущи только ей и которые будут отличать данную модификацию от базового продукта и других модификаций. С другой стороны, возможен вариант, когда для разных версий одной и той же модификации требования будут

неизменными, потому что каждая версия модификации будет являться очередной попыткой удовлетворить этот же набор требований (будем надеяться, что при этом система будет только улучшаться).

В предыдущих разделах мы использовали термин «релиз» и, чтобы не запутывать читателей, необходимо пояснить разницу между версией продукта и его релизом. Отличие состоит в том, что релиз это версия, которая была доставлена (доступна) пользователям. Понятно, что в силу целого ряда причин, далеко не каждая версия продукта доходит до пользователей.

Планирование жизненного цикла модификаций и их версий для каждого продукта является дополнительной организационной задачей, для решения которой может применяться механизм «ворота стадий». Разработка различных модификаций одного продукта может проходить параллельно во времени, но при этом необходимо поддерживать как минимум хотя бы одну версию каждой модификации, пока продукт находится в коммерческой эксплуатации.

Этапы разработки требований и производства для компании-производителя во многом схожи с аналогичными этапами, описанными ранее в отношении компании-покупателя и компании-поставщика. Основное же отличие заключается в наличии большого объема общей информации, который повторно используется при разработке различных версий и модификаций одного продукта. Это в значительной степени усложняет управление требованиями, поскольку необходимо четко представлять и понимать, как реализация общих для каждой версии и модификации требований накладывается друг на друга во времени.

Управление требованиями еще в большей степени усложняется в тех случаях, когда общие требования принадлежат нескольким существующим версиям и модификациям, поскольку это ведет к дополнительным проблемам в управлении изменениями (см. ниже).

8.5.2 Контроль за ходом выполнения работ

Для контроля за ходом выполнения работ в организациях-производителях использует точно такой же механизм, как и в организациях других типов. При использовании метода «ворота стадии» процесс планирования работ должен предусматривать получение такой информации или данных, которые должны в обязательном порядке присутствовать в конце стадий. Степень готовности (наличие) этой информации и данных, в свою очередь, может являться мерилом прогресса работ по проекту. В этом случае ход работ может быть измерен (оценен), например, с помощью следующих критериев:

- созданы ли новые артифакты, которые могут служить объектами трассировки (*например*, элементы решения, связанные с пользовательскими требованиями, или элементы спецификаций, соответствующие системным требованиям);
- заполнены ли значения атрибутов;
- определены ли статусы проверки;
- существуют ли необходимые связи (трассировка) от одних наборов данным к другим (*например*, связи между пользовательскими и системными требованиями; или связи между системными требованиями и элементами спецификации; или в целом - связь от требований к стратегии проверки и далее к результатам выполнения тестов).

Набор этих критериев, выраженный в процентном отношении достигнутых показателей к планируемым, формирует прекрасную систему не только показателей качества получаемых данных, но и степени прогресса работ в рамках конкретной стадии.

8.5.3 Изменения

Как упоминалось ранее, главную сложность в управлении изменениями в организациях-производителях, создает ситуация, при которой несколько модификаций одного продукта имеют общие для них требования, а предлагаемое изменение затрагивает одно или несколько этих требований. В этом случае появления запроса на изменение заставляет искать ответы на следующие вопросы:

- должно ли вносимое изменение затрагивать все модификации продукта?;
- когда данное изменение должно быть внесено в каждую из модификаций?

Нетрудно угадать наиболее распространенный ответ - изменение должно быть внесено во все модификации, но не одновременно!
Такая ситуация требует введения в диаграмму переходов дополнительного состояния (см. рис. 8.6), потому что в этом положении изменение должно быть «примерено» по отношению к каждой модификации с тем, чтобы первичный запрос на изменение мог бы считаться отработанным до конца.

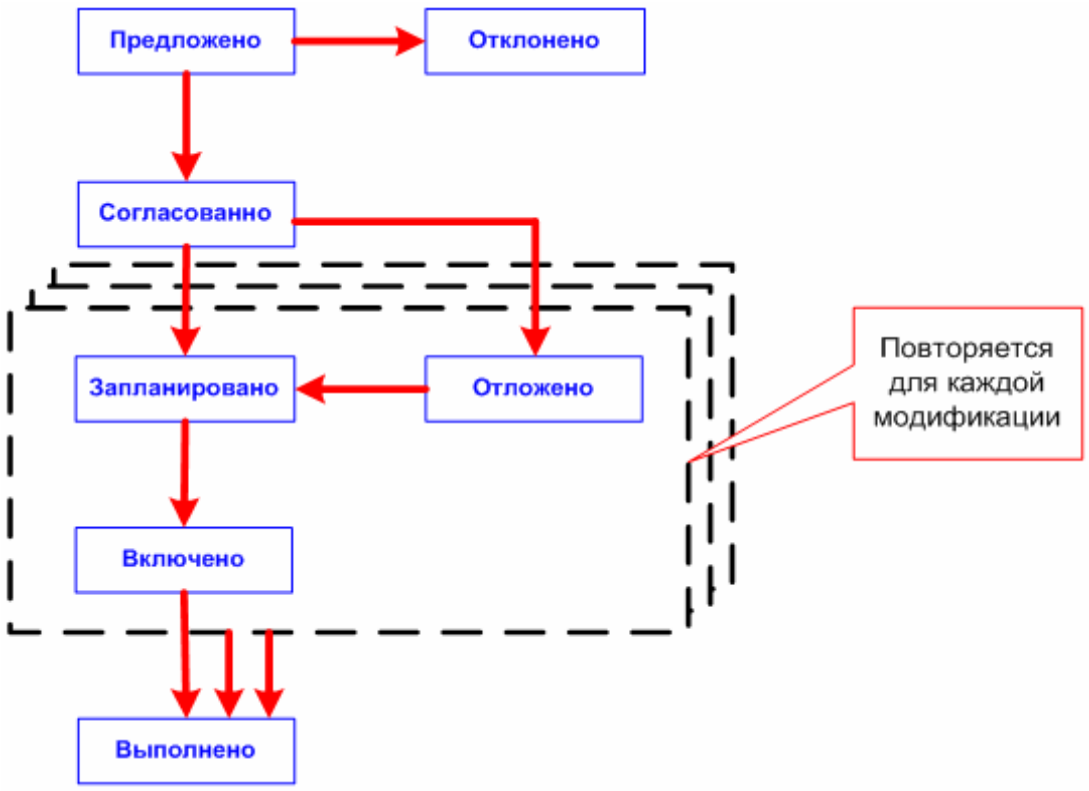


Рис. 8.6 Диаграмма переходов состояний для управления изменениями требований для модификаций.

Из рис. 8.6 видно, что для каждой модификации вносимые изменения должны иметь свои собственные значения состояний «Запланировано», «Отложено», «Включено». При этом значение «Выполнено» состояние первичного изменения может принять только после отработки его по всем модификациим продукта.

В заключение необходимо отметить, что организациями-производители решают те же задачи, что и организации-покупатели и организации-поставщики. Но в дополнение к этому, организации-производители должны прилагать немало усилий для управления портфелями продуктов и уделять больше внимания экономическим аспектам, таким как увеличение доли рынка и возврат инвестиций.

8.6 Заключение

Информация в заключении специально сгруппирована под теми же заголовками, что и последовательность материала, изложенного в данной главе.

8.6.1 Планирование

Основой планирования должны служить результаты, которые предполагается получить. Имея информацию о конечных результатах, можно планировать дальнейшие мероприятия, необходимые для их достижения.
Конечными результатами могут быть:

- различные объекты информации (*например*, список заинтересованных сторон, пользовательские требования, системные требования, элементы спецификации или решения);
- атрибуты, относящиейся к соответствующим информационным объектам;
- связи между объектами информации, необходимые для организации трассировки, стратегией проверки, контроля изменений и т.п.;
- критерии для рецензирования и анализа качества информационных объектов и их атрибутов;
- достижение определенных состояний объектов (*например*, последовательная смена значений статуса рецензируемого требования по результатам его анализа).

Перед выполнением любой работы она обязательно должна быть согласована и утверждена внутри компании.
Для организации-покупателя и организации-производителя метод «ворота стадии» вполне пригоден для контроля за выполнением хода работ и достижением соответствующих финансовых и\или экономических показателей, которые компания желает достичь.
В компании-поставщике должна быть обязательно авторизована работа по подготовке коммерческого предложения, поскольку эта деятельность тесно связана с бюджетом, выделяемым из собственных средств. Разрешением же на разработку обычно является факт заключения контракта с заказчиком.

Нормой - особенно в отношении разработки абсолютно новых систем - должна стать последовательная (итерационная) разработка. Это, естественно, ведет к использованию

механизма релизов, версий и модификаций, что, в свою очередь, значительно облегчает планирование.

8.6.2 Контроль за ходом выполнения работ

Основной концепцией контроля за ходом выполнения работ является регулярная проверка степени соответствия текущего результата запланированному. Постоянное сравнение достигнутых результатов с планируемыми параллельно с контролем затраченного времени и ресурсов дает возможность оценивать текущее выполнение плана. Игнорирование же результатов, а использование лишь контроля времени и ресурсов, формирует весьма искаженное представление о ходе работ по проекту.

8.6.3 Изменения

Наиболее критичным аспектом изменений является их воздействие на разрабатываемую систему, а, следовательно, и на план разработки. Оценить влияние предлагаемых изменений можно только в том случае, если все текущие результаты доступны и имеют актуальный (постоянно обновляемый) статус. В этом смысле трудно переоценить значение связей (трассировки) между входящей и производной информацией.

В зависимости от категории и масштабов изменения решение о его принятии может привести к существенному изменению плана. Зачастую изменения приводят к появлению дополнительных релизов, версий или модификаций.

9 DOORS: Средство управления требованиями

Тут нет ничего особенного.
Все что нужно делать человеку, это в нужный момент
нажимать нужные клавиши, а инструмент будет играть сам.
Иоганн Себастьян Бах (Johann Sebastian Bach),
композитор, 1685 – 1750

9.1 Введение

Для поддержки процесса управления требованиями аналитикам и руководителям необходим хороший инструментарий. В настоящее время на рынке нет недостатка в продуктах, предназначенных для управления требованиями. Но в данной главе мы рассмотрим лишь один из них – Telelogic DOORS (версия 8.0[9]).
DOORS (Dynamic Object Oriented Requirements System – динамическая объектно-ориентированная система для работы с требованиями) является на сегодняшний день лидирующим на рынке продуктом, который используется десятками тысяч специалистов во всем мире. Первоначально продукт был разработан компанией QSS Ltd (Оксфорд), а впоследствии приобретен, поддерживается и продолжает развиваться компанией Telelogic.

DOORS является многоплатформенной системой управления требованиями, которая предназначена для хранения и управления информационными объектами различных типов, для создания связей между этим объектами, а также – что немаловажно - для их анализа. DOORS обеспечивает поддержку отраслевых и корпоративных стандартов разработки. DOORS также обеспечивает поддержку коллективной работы и позволяет различным группам специалистов, взаимодействуя друг с другом, находить правильные решения для удовлетворения требований бизнеса. Более того, DOORS позволяет делать это максимально эффективно. DOORS обладает мощным, интуитивно-понятным навигационным механизмом, обеспечивающим удобство работы с требованиями.

Для иллюстрации возможностей DOORS мы продолжим рассматривать здесь уже знакомый нам по предыдущим главам пример с парусной лодкой.

9.2 Необходимость управления требованиями

Сегодня аналитикам и системным архитекторам просто необходимо иметь эффективное средство управления требованиями для выработки высокотехнологичных решений. Управление требованиями представляет из себя процесс, в рамках которого происходит первичное накопление, формирование, анализ и последующий контроль за реализацией потребностей заинтересованных сторон, а также дальнейшее управление изменениями информации на протяжении всего жизненного цикла проекта. Продукты также становятся

[9] *прим. переводчика:* В оригинале книги рассматривается версия 7.1, но мы приняли решение отступить от оригинала и использовать для изложения примеры, выполненные на базе последней версии DOORS – 8.0.

все более и более сложными, давно перешагнув ту черту, за которой один единственный человек мог бы не только полностью понимать, что собой представляет вся система в целом, но и в тоже время знать досконально все составляющие ее части.

Структурирование требований является наилучшим способом их организации, позволяя более эффективно управлять ими во избежание дублирований или пропусков информации. В каком-то смысле управление требованиями, с одной стороны, выглядит как процесс передачи информации, а с другой стороны – сами требования выполняют роль средства общения. В связи с этим очень важно, чтобы требования корректно передавали смысл и правильно связывались между собой для более качественного взаимодействия, обеспечивая уверенность в том, что совместная работа команды эффективна, риски проекта снижены, а выполнение проекта движется в правильном направлении и отвечает сути поставленных бизнесом целей.

Эффективное управление требованиями обеспечит компании поставку на рынок «правильного продукта» в нужное время и в рамках запланированного бюджета.

9.3 Архитектура DOORS

В DOORS все требования и связанная с ними информация храняться в центральной базе данных. База данных DOORS должна быть доступна и поддерживаться в актуальном состоянии на протяжении всего жизненного цикла приложения.

Вся информация в базе данных DOORS храниться в виде модулей (*module*) (см. рис. 9.1). Проекты (*project*) и папки (*folder*) используются в DOORS для более организованного представления модулей. Проект, по сути, является той же папкой, но предназначен для хранения всей информации, относящейся к данному проекту или подпроекту.

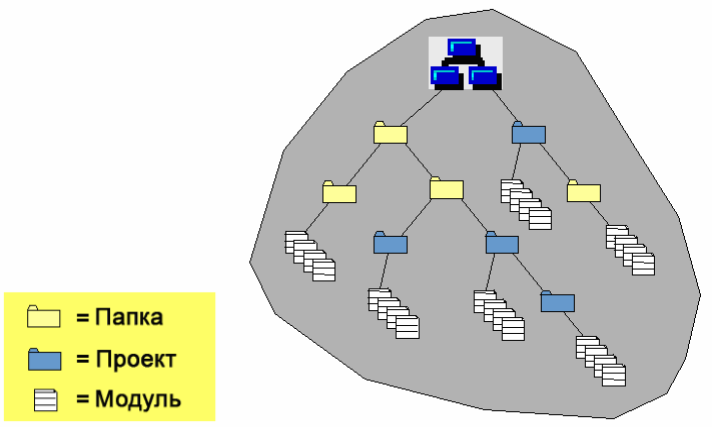


Рис. 9.1 Структура базы данных DOORS.

Папки в системе DOORS предназначены для организации информации и напоминают каталоги файловой системы. Папки могут содержать другие папки, проекты и модули. Каждая папка имеет собственное имя и описание (характеристику). Степень

манипулирования информацией, содержащейся в папке, может быть ограничена для каждого пользователя и регулируется правами доступа.

Проекты в системе DOORS используются группами специалистов для организации данных, относящихся к их совместной работе. Проект должен (рекомендуется) содержать всю информацию, касающуюся требований, спецификаций, разработки, тестирования, выпуска и поддержки приложения. В рамках проекта обеспечивается возможность назначать права доступа ко всей информации, содержащейся в данном проекте, делать резервные копии данных (в отличие от папок), а также экспортировать блоки данных в другие базы данных DOORS.

Модули в системе DOORS являются теми контейнерами, которые содержат различные наборы данных. Существуют три типа модулей:

- *formal (формальный)* – наиболее часто используемый тип модуля, используемый для хранения структурированных наборов идентичной информации;
- *descriptive (описательный)* – модуль, используемый для хранения неструктурированной первичной информации (переписка, заметки сделанные в ходе интервью);
- *link (связи)* – модуль, содержащий взаимосвязи и информацию о них между элементами данных.

Пользовательский интерфейс DOORS во многом напоминает интерфейс Windows Explorer (Проводник) и позволяет легко и удобно просматривать данные в системе.

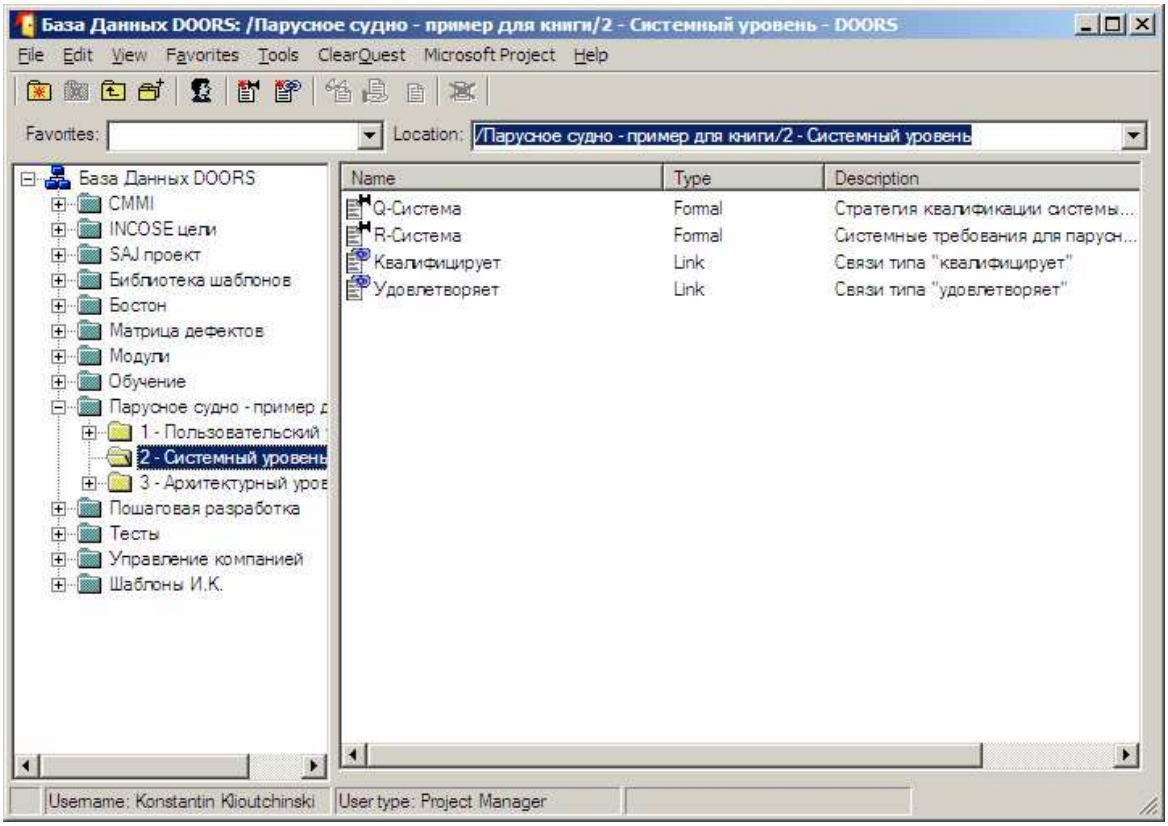


Рис. 9.2 Окно базы данных DOORS.

9.4 Проекты, модули и объекты

9.4.1 Окно базы данных DOORS

Окно базы данных DOORS дает возможность пользователю видеть и управлять структурой базы данных системы. На рис. 9.2 приведен пример окна базы данных DOORS, в левой стороне которого отображается структура всей базы, а с правой – отображается содержимое той папки, которая слева выделена курсором.

В DOORS предусмотрена возможность изменять названия и описания существующих папок и проектов. При необходимости изменить или реорганизовать структуру базы данных папки и проекты можно перемещать. В DOORS предусмотрена также возможность копировать, вырезать и вставлять проекты и папки, используя команды *Copy*, *Cut* и *Paste*.

9.4.2 Формальные модули

Для создания в DOORS нового формального модуля необходимо выполнить следующую последовательность команд из главного меню: **File ► New ► Formal Module**, как это показано на рис. 9.3.

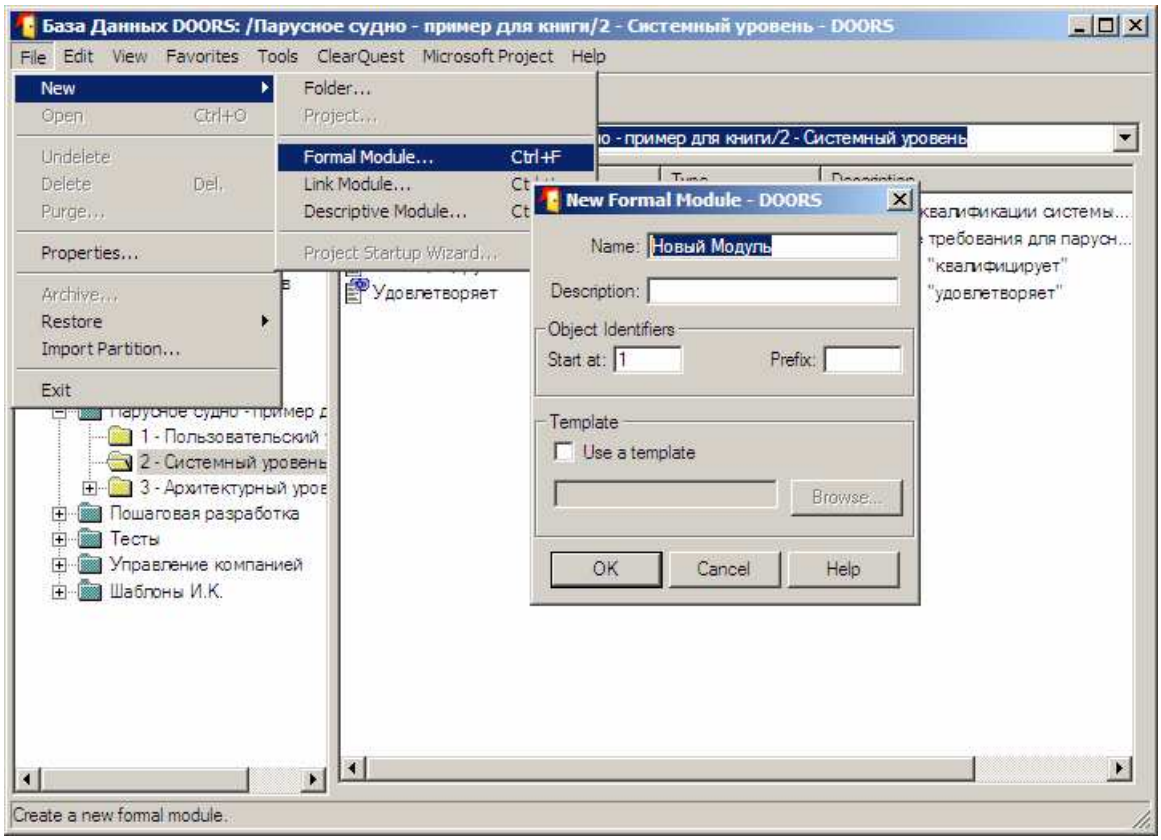


Рис. 9.3 Создание нового формального модуля.

Далее необходимо ввести имя модуля в поле *Name* (обязательно!) и описание в поле *Description* (рекомендуется). Поскольку для каждого нового объекта внутри модуля система автоматически будет создавать уникальный идентификационный номер, то в этом же окне можно задать число, с которого будет начинаться нумерация объектов модуля (*Start at*:). Помимо номера идентификатор объекта может содержать префикс (*Prefix*), который также можно задать при создании модуля. Префикс используется для более наглядного отображения содержимого модуля; например, для модуля, содержащего пользовательские требования, можно было бы задать префикс **ПТ**. Если для каждого формального модуля, входящего в конкретный проект, задать свой уникальный префикс, то простым способом мы получим уникальную идентификацию всех объектов всех модулей внутри данного проекта. В этом случае можно (и это проще) ссылаться на идентификатор объекта без указания названия модуля и всем будет понятно, какой именно модуль имеется в виду.

Если же открыть существующий формальный модуль, то, по умолчанию, слева будет окно структуры данных модуля (Module Explorer), а справа окно с данными, содержащимися в модуле (см. рис. 9.4).

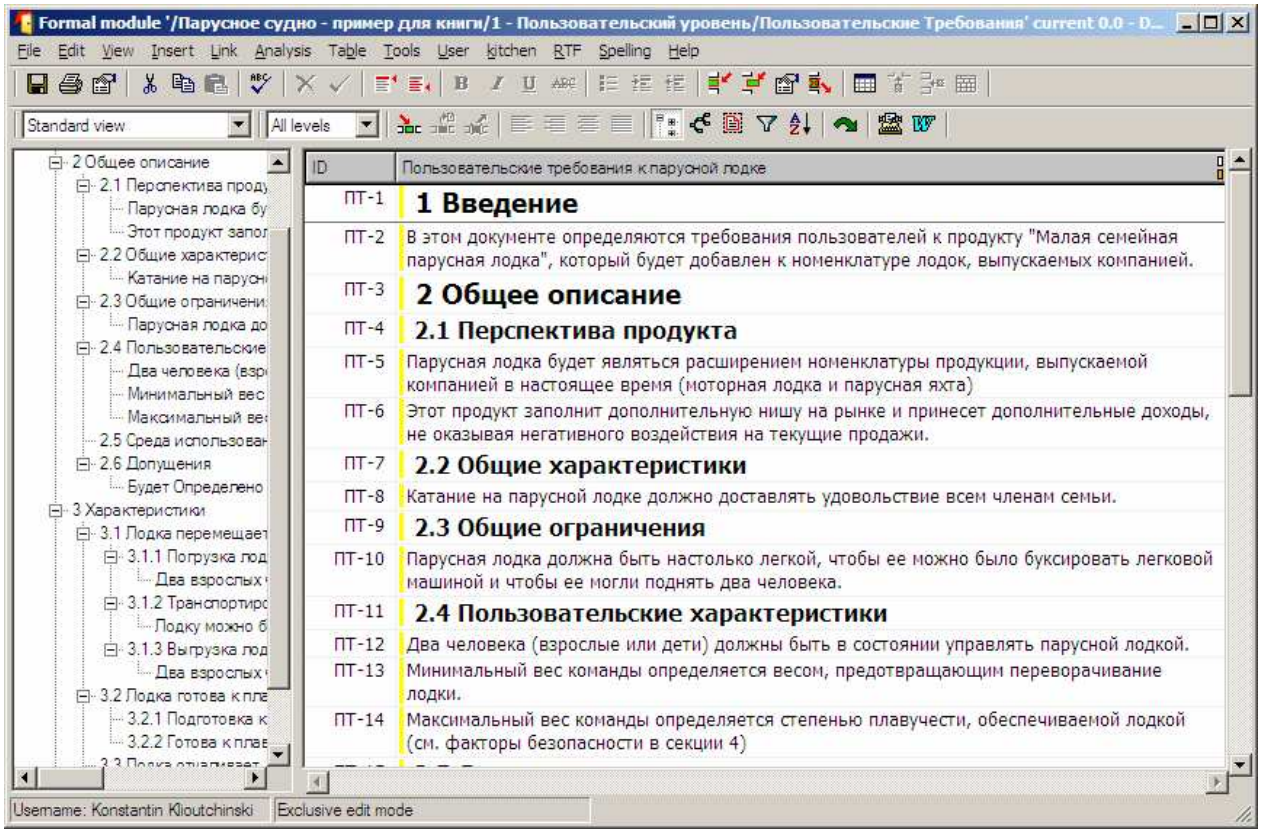


Рис. 9.4 Вид формального модуля (по умолчанию).

Структура модуля полностью отображает структуру информации, содержащейся в модуле, и при необходимости позволяет легко перемещаться в нужное место. При этом, в окне структуры модуля возможно производить все те же манипуляции, что и в Проводнике (Windows Explorer), т.е. свертывать (-) и разворачивать (+) элементы структуры.

В правой части окна открытого модуля отображаются данные, которые в нем содержатся. По умолчанию здесь открываются две колонки – идентификатор *ID* и текстовая колонка, заголовком которой является описание модуля из поля *Description*. В колонке *ID* содержатся уникальные идентификаторы объектов, автоматически присваиваемые системой каждому объекту при его создании. DOORS использует этот идентификатор для того, чтобы поддерживать связь между объектом и ассоциированной с ним любой другой информацией (*напр.*, его атрибутами), а также с другими объектами. Текстовая колонка отображает содержимое модуля в формате документа, т.е. показывает комбинацию нумерации заголовков, текста заголовков и текста требований (объектов).

DOORS дает возможность отображать данные в различном формате (см. рис. 9.5). В формате *Standard View* объекты всех уровней просто отображаются в виде документа. При этом пользователь, при желании, может ограничить объем отображаемой информации, выбрав другой вид, например, формат *Outline View*. В этом случае DOORS отобразит только заголовки, скрыв другие детали объектов. Этот вид похож на раздел книги, который называется «Оглавление». Вид *Explorer View* весьма удобен для отображения иерархической структуры модуля и поиска нужного объекта в модуле.

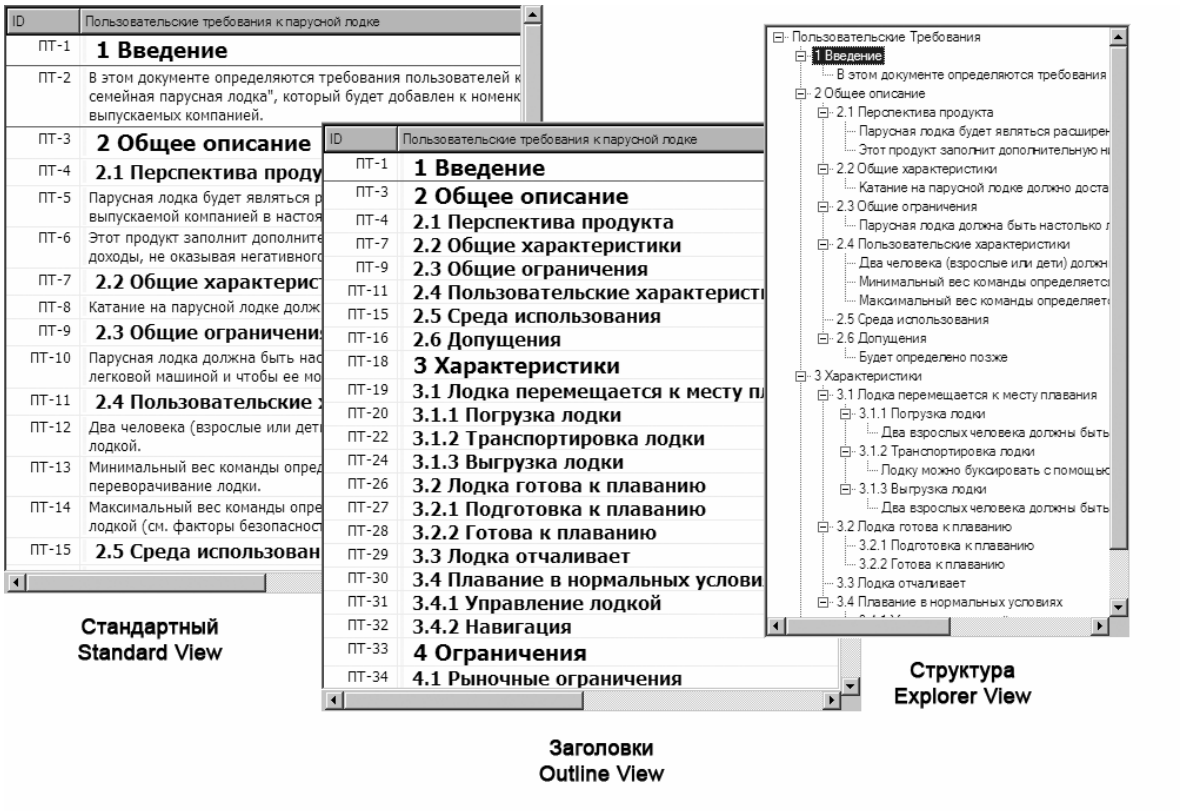


Рис. 9.5 Режимы отображения формального модуля.

Графический режим *Graphics Mode* позволяет отобразить информацию, содержащуюся в модуле, в виде иерархического дерева (см. рис. 9.6). Это значительно облегчает навигацию и поиск, если необходимая информация «спрятана» в больших объемах данных. В качестве названия элемента иерархии в этом случае используется заголовок объекта (*Object Heading*) и сокращенный текст самого объекта (*Object Text*).

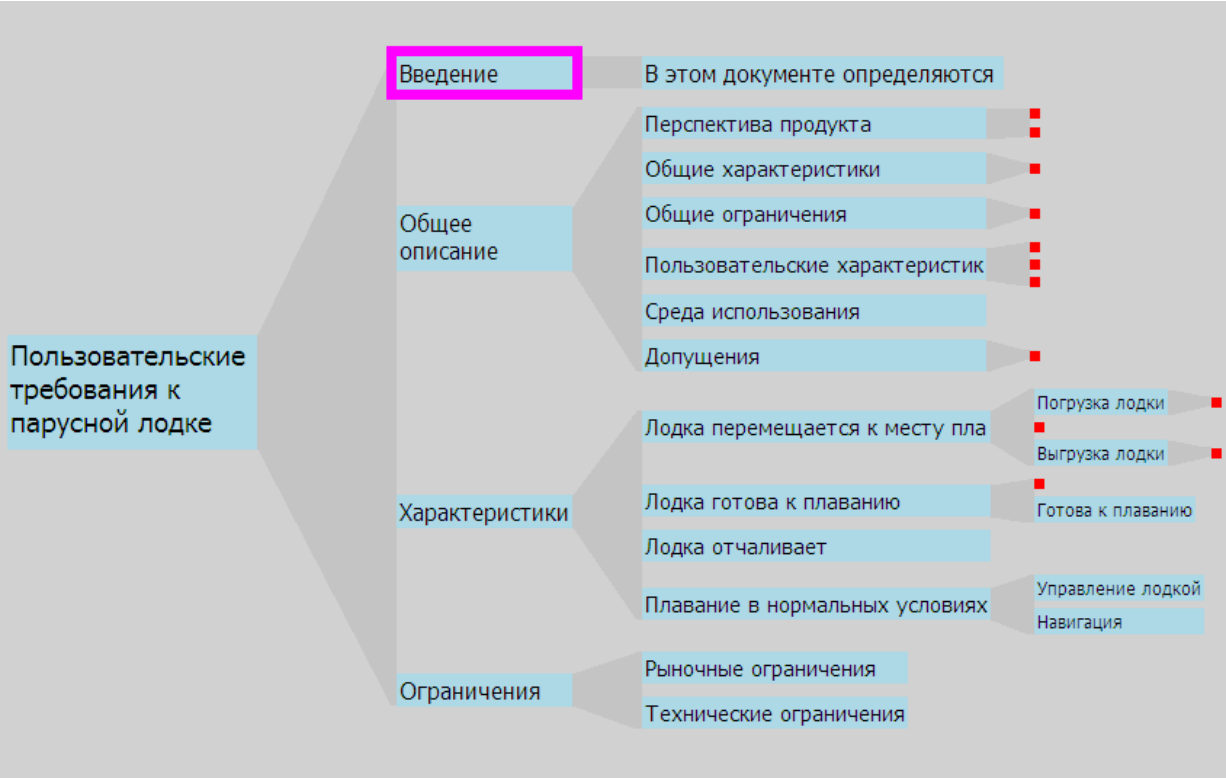


Рис. 9.6 Графический режим.

9.4.3 Объекты

Как уже упоминалось в предыдущих разделах, данные в формальных модулях хрянятся в виде объектов. Объектом может быть некоторый текст, графическое изображение или даже электронная таблица, созданная с помощью другого приложения. Стандартный вид формального модуля содержит две колонки и несколько дополнительных индикаторов, о которых мы расскажем ниже.

На рис 9.7 показана информация, отображаемая в окне открытого модуля. Первая колонка показывает идентификатор объекта (*Object Identifier*), который автоматически присваивается DOORS. Идентификатор объекта состоит из двух частей:

- префикс (обычно это аббревиатура, характерная для данного набора требований);
- уникальный номер, присваиваемый DOORS.

Уникальный номер представляет собой целое число, присваивается системой автоматически и исключительно последовательно (1, 2, 3 и т.д.) каждому вновь создаваемому объекту и служит ключевой характеристикой объекта. Номер объекта уникален (единственен) внутри данного модуля.

Вторая колонка называется главной (*Main*) или текстовой (*Text*) и содержит сочетание трех различных атрибутов:

- номер раздела (например, 1, 2.1, 3.2.3), который определяет положение объекта в структуре модуля;
- заголовок объекта;
- текст объекта.

Номера разделов присваиваются только тем объектам, которые имеют заголовок.

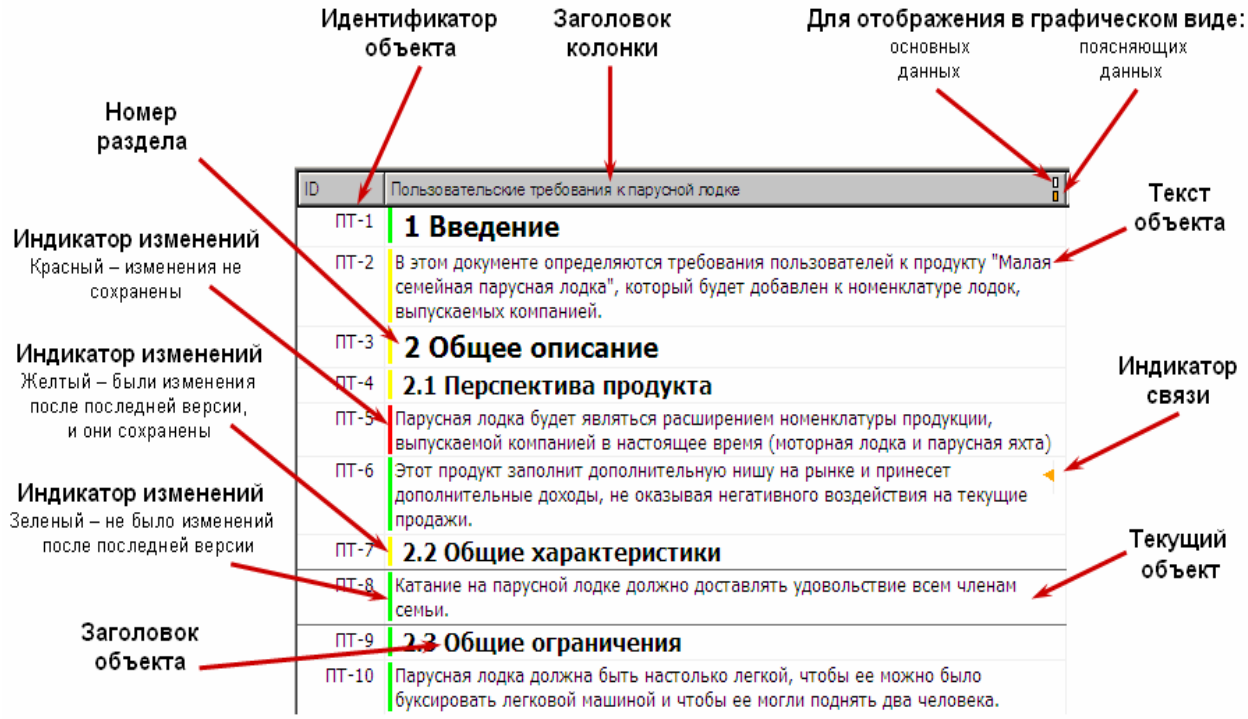


Рис. 9.7 Отображаемая информация.

Объект, выделенный сверху и снизу более темными горизонтальными линиями, носит название «текущего объекта» (*Current Object*). Большинство функций в системе DOORS, выполняемых в рамках данного модуля (*напр.*, создание нового объекта, вставка объекта, перемещение объекта и т.д.), выполняются по отношению к текущему объекту.

Зеленые, желтые и красные полоски с левой стороны текстовой колонки являются индикаторами изменений (*Change Bars*). Зеленая полоска обозначает, что объект ни разу не изменялся с тех пор, когда данный вариант модуля был зафиксирован в качестве очередной версии (baseline). Желтая обозначает, что со момента последнего сохранения версии модуля в объект вносились какие-либо изменения. Красная же полоска обозначает, что изменения объекта, выполненные в текущей сессии, пока еще не сохранены в базе данных.

Оранжевый и темно-коричневый треугольники с правой стороны текстовой колонки показывают наличие связей данного объекта с другими объектами (*Link Indicator*). Оранжевый треугольник, направленный влево (внутрь объекта), обозначает наличие входящих связей, а темно-коричневый треугольник (не показан), направленный вправо (наружу из объекта), обозначает наличие исходящих связей.

Древовидная структура отображения информации в формальном модуле DOORS обеспечивает очень простой, но весьма мощный и эффективный метод разработки и

управления требованиями. Поскольку требования очень часто имеют иерархическую структуру, то графическое отображение данных является также весьма удобным и полезным.

Создание новых объектов в DOORS выполняется достаточно легко (см. рис. 9.8) – новый объект просто вставляется на одну из двух возможных позиций относительно текущего объекта:

- новый объект создается на том же уровне иерархии, что и текущий объект (команды **Insert ► Object**), или
- новый объект создается как первый дочерний объект по отношению к текущему объекту (команды **Insert ► Object Below**).

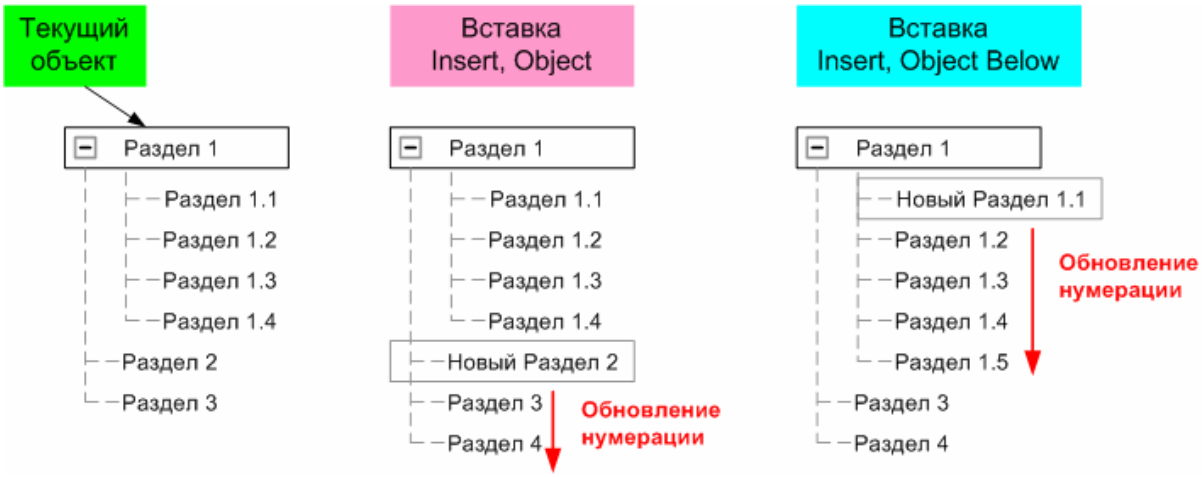


Рис. 9.8 Создание объектов.

DOORS также предлагает мощные средства для редактирования структуры модуля. Например, древовидная структура модуля может быть скорректирована с помощью функций вырезки и вставки (*Cut* и *Paste*).

Рис. 9.9 Вырезка и вставка объектов.

Операция «вырезать» (*Cut*) убирает текущий объект и все его дочерние (!) объекты из структуры модуля. Это операция вызывает автоматическое обновление нумерации заголовков всех объектов в модуле так, чтобы устранить образовавшийся разрыв в структуре. Вырезанный объект может быть затем вставлен (*Paste*) либо после текущего объекта на том же уровне иерархии, либо в качестве первого дочернего объекта по отношению к текущему (см. рис. 9.9).

9.4.4 Графические объекты

DOORS поддерживает OLE-технологию (Object Linking and Embedding) для вставки в текстовый атрибут любых объектов, в том числе, и графических. Такие объекты вставляются в модуль точно так же, как это делается, например, в MS Word. С помощью этой технологии в документ с требованиями можно вставлять рисунки, диаграммы, графики, документы, электронные таблицы, а также множество другой информации, необходимой для пояснения того, что утверждается в требовании.

9.4.5 Таблицы

Зачастую требования, или связанная с ними информация, отображаются в виде таблицы.

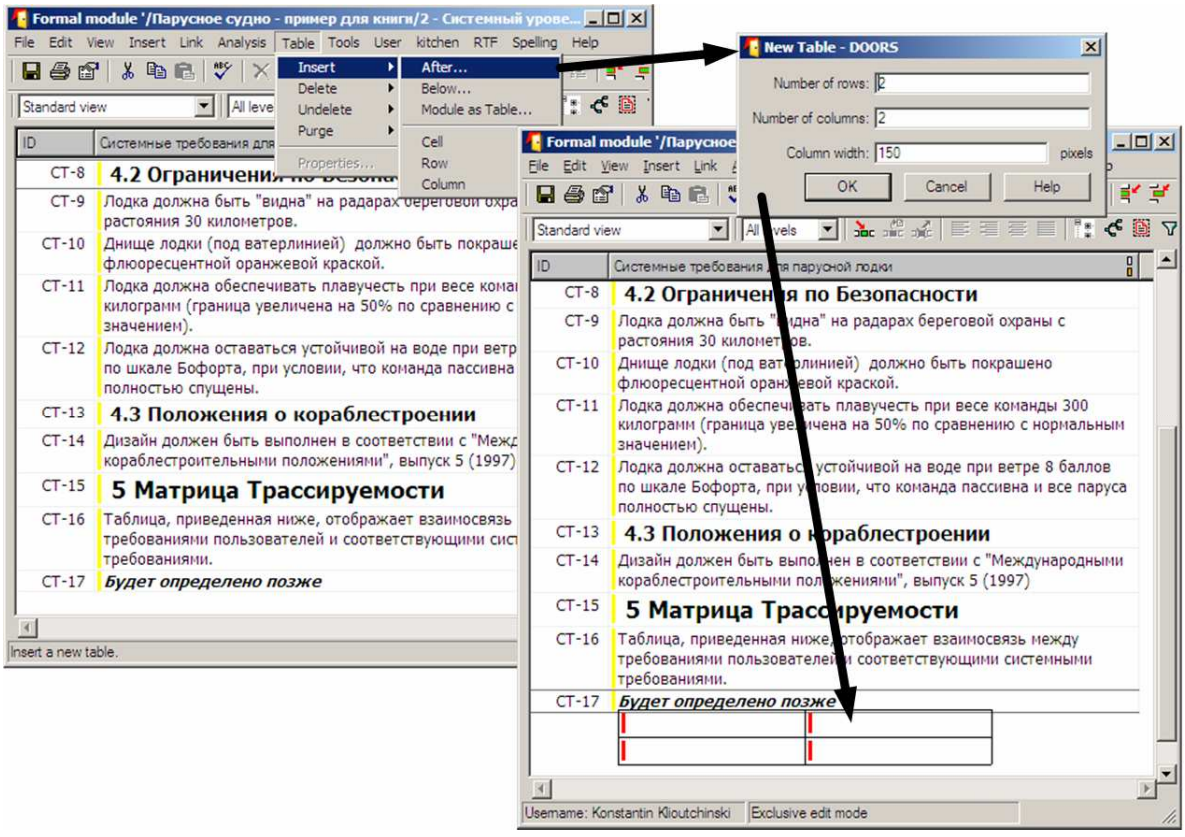


Рис. 9.10 Создание таблицы.

В этом случае таблица может быть создана либо до/после выбранного объекта, но на одном иерархическом уровне с ним, либо просто в пустом модуле на самом верхнем уровне. При создании таблицы необходимо задать количество колонок и строк, как это показано на рис. 9.10.
Следует заметить, что каждая ячейка таблицы, создаваемой в DOORS, представляет собой отдельный объект (в отличие от таблицы Excel, которая может быть вставлена в модуль как OLE-объект). Таблица может быть удалена, только в том случае, если нет связей ни с ней, ни с одной из ее ячеек.

9.5 История изменений и версии

9.5.1 История изменений

DOORS сохраняет историю изменений всех модулей, объектов и их атрибутов.

Запись любого изменения отобразит автора изменения, дату и время изменения, а также состояние объекта и его атрибутов до и после внесенного изменения. История модуля содержит все события, которые имели место по отношению к модулю.

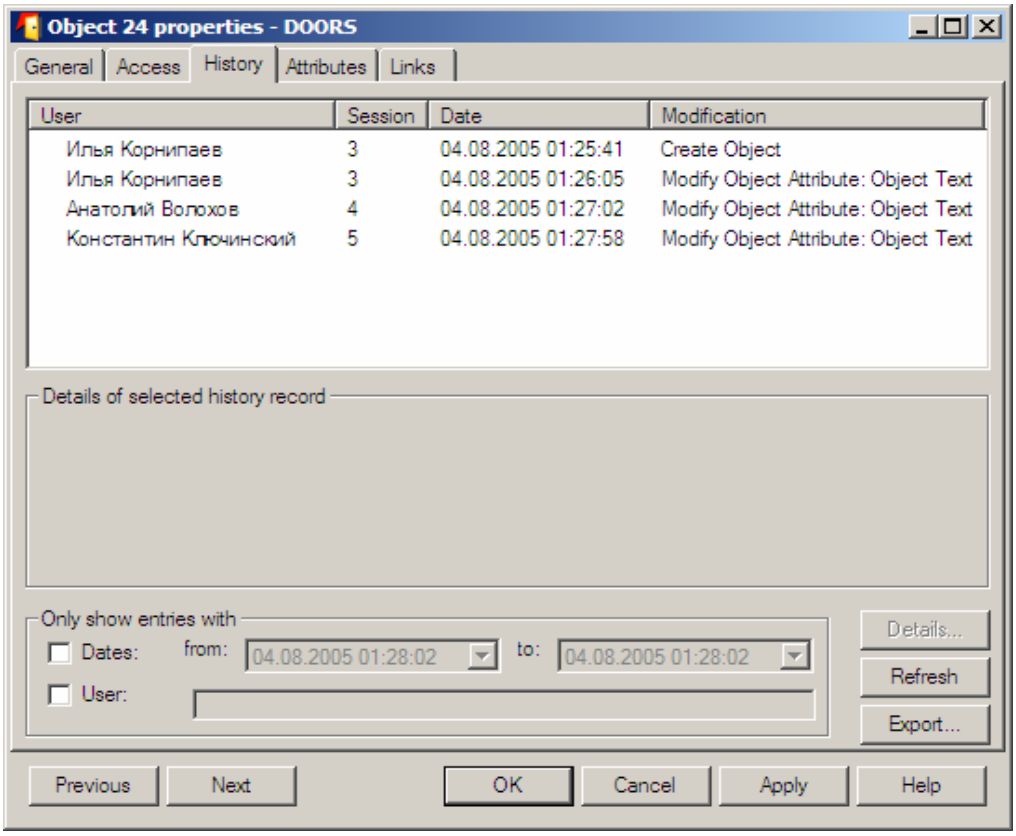


Рис. 9.11 Окно истории изменений объекта

Историю любого объекта можно открыть, либо кликнув мышкой на индикатор изменений (см. рис. 9.7), либо через главное меню в окне модуля. Пример окна истории изменений объекта приведен на рис. 9.11.

9.5.2 Версии

Версия (baseline) - это «замороженная» копия модуля.

Версии создаются обычно по завершению определенных стадий проекта (*напр.*, одна из версий создается непосредственно перед рецензированием, другая - сразу после того, как замечания, сделанные в ходе рецензирования, внесены в требования). Такой подход позволяет в любой момент получить версию документа, характерную для определенного этапа проекта.

Каждой зафиксированной версии модуля в DOORS присваивается номер и название. Версии модуля существуют в DOORS только для просмотра (read-only) и их нельзя редактировать.

После создания версии вся история изменений объектов и их атрибутов сохраняется (остается) вместе с зафиксированной версией, в то время как для текущей версии модуля история изменений объектов отсутствует (становиться пустой) и вновь начинается с нуля. Таким образом, история жизни модуля сохраняется в виде серии версий.

9.6 Атрибуты и виды

9.6.1 Атрибуты

Атрибуты являются тем средством, которое позволяет «привязывать» к модулям и объектам информацию, относящуюся к ним.

Атрибуты модуля используются для хранения информации о самом модуле, например таких параметров как, автор модуля, идентификационный номер модуля в системе, статус рецензирования модуля и т.д. По аналогии, атрибуты объекта используются для хранения любой дополнительной информации о данном объекте.

Существуют два типа атрибутов: системные атрибуты и атрибуты, определяемые пользователем. Системные атрибуты автоматически создаются системой для хранения важной системной информации (*напр.*, даты и времени создания модуля или объекта, автора создания объекта или или автора вносимого изменения и т.п). В то время как атрибуты, определяемые пользователем, могут создаваться и использоваться для поддержания собственного уникального процесса управления требованиями в каждой конкретной организации.

По умолчанию в DOORS уже заложены и могут использоваться различные стандартные типы атрибутов, называемых также б*азовыми типами*. Это дает возможность создавать и определить атрибуты такие, например, как целое число (*integer*), вещественное число (*real*), дата (*date*), строка (*string*), текст (*text*), имя пользователя (*user name*). Но при этом пользователь также может создавать и свои собственные типы атрибутов.

Любая информация, относящаяся к атрибутам, может быть легко отражена в модуле с помощью создания в нем новых колонок.

Таким образом, информацию, содержащуюся в атрибутах, можно просматривать и редактировать на экране, а также выводить на принтер. Чтобы не перегружать пользователя лишней информацией (поскольку объект может содержать значительное число атрибутов), в DOORS имеется возможность выводить на экран только те атрибуты объекта, которые необходимы для работы. Для изменения порядка расположения колонок на экране используется технология drag-and-drop – достаточно просто, «взяв» колонку за заголовок «перетащить» ее в нужное место на экране.

9.6.2 Виды

Для того, чтобы просматривать одну и ту же информацию с разных точек зрения, в DOORS предусмотрена категория, называемая «виды» (Views), т.е. предусмотрена возможность манипулировать информацией, отображающейся на экране (и принтере).

Пользователь может создавать неограниченное число отличающихся друг от друга видов отображения информации, которые будут храниться в модуле. При создании видов пользователь имеет возможность не только манипулировать расположением колонок с атрибутами, но и задавать критерии отображаемой информации, используя фильтрацию (*напр.*, показывать только такие объекты модуля, у которых атрибут «Приоритет» имеет значение «Высокий»). И тогда вид будет содержать только ту информацию, которая в данный момент удовлетворяет заданным параметрам.

9.7 Связи и их анализ

Doors дает возможность создавать связи между объектами, а также предоставляет средства для их анализа.

9.7.1 Связи

В DOORS предусмотрена возможность связывать объекты между собой. В терминах DOORS связь называтся *Link*.

Одним из важных свойств связи является ее направленность. Все связи в DOORS направлены от источника к цели. Использование связей позволяет наглядно представлять информацию не только в виде иерархической (древовидной) структуры, но и в виде сетевой структуры. При этом, несмотря на то, что связи в DOORS однонаправленные, система дает возможность переходить по связям между объектами в обоих направлениях. Таким образом, DOORS позволяет легко выполнить анализ влияния изменения, вносимого в один документ, на связанные объекты во всех других документах, а также предоставляет возможность разыскать ту информацию, которая послужила причиной для принятия того или иного решения.

DOORS предлагает несколькими способов создания связей и последующей работы с ними. Индивидуальная связь между двумя объектами может быть создана с помощью

технологии drag-and-drop (обычно так создаются связи между двумя объектами в одном или разных модулях). Для создания наборов (блоков) связей предпочтительнее воспользоваться другим способом. Например, функция *copy and link* позволяет скопировать необходимый набор объектов и тут же связать каждую копию с ее оригиналом.

Наличие связей отображается в основной колонке стандартного вида формального модуля в виде небольших треугольников с правой стороны. При этом треугольник оранжевого цвета, направленный влево (внутрь), обозначает наличие входящих в данный объект связей, а треугольник темно-коричневого цвета, направленный вправо (наружу), обозначает наличие исходящих из этого объекта связей.

9.7.2 Отчеты о связях

В DOORS существует множество способов создания отчетов, отражающих связи между объектами, как для вывода на экран, так и для последующей печати на принтере.

Наиболее простым способом получения экранного отчета о связях (или, как еще говорят, - о трассировке) является использование анализатора связей (в главном меню нужно выбрать **Analysis ► Traceability Explorer**), который дает возможность пользователю в одном окне видеть связи сразу между многими документами (см. рис. 9.12).

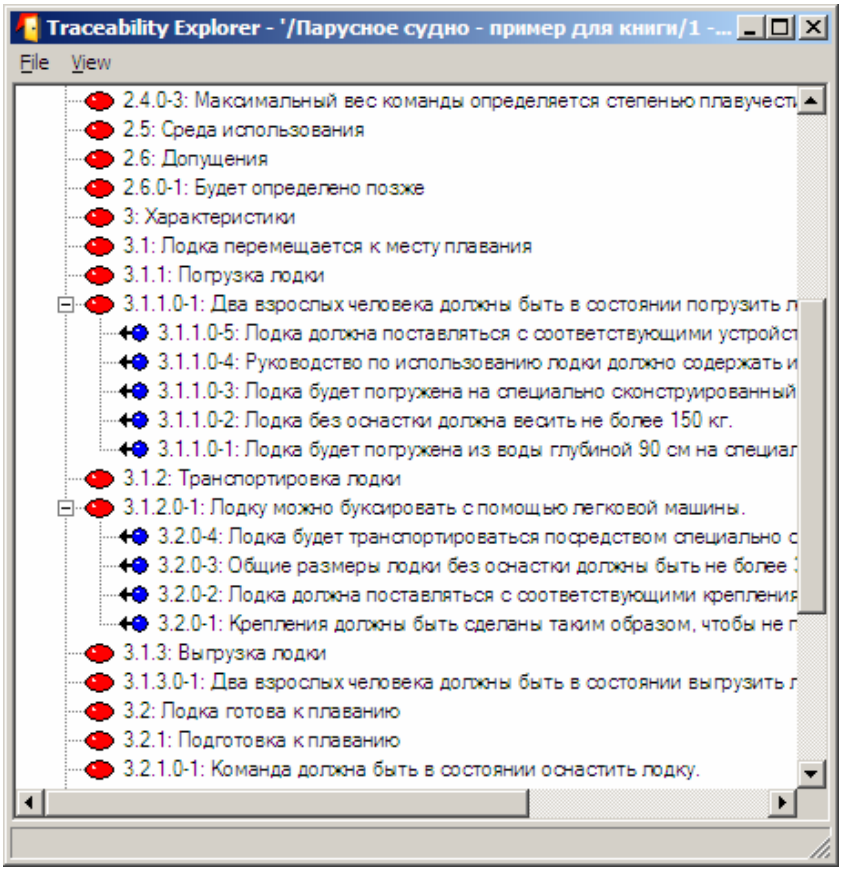


Рис. 9.12 Анализатор связей (Traceability Explorer).

Другим способом создания отчета, показывающего связи между объектами, является создание такого вида (View), в котором связи отображаются в специально созданных для этого колонках трассировки. (Такой отчет может быть выведен и на принтер).
Для создания в DOORS такого вида необходимо воспользоваться пунктом меню **Analysis ► Wizard**, который вызовет программу-мастера, позволяющую пошагово определить - информацию о каких именно связях и атрибутах объектов необходимо отобразить. Колонки трассировки, в которые выводится эта информация, имеют динамическую природу и автоматически обновляются каждый раз, как только создается новая связь или изменяются данные на другом конце существующей связи. Использование этого подхода позволяет «собирать» информацию из различных модулей в единый отчет, с которым можно работать и на экране, и который легко распечатывается на принтере.

На рис. 9.13 показан пример вида, содержащего колонки трассировки.

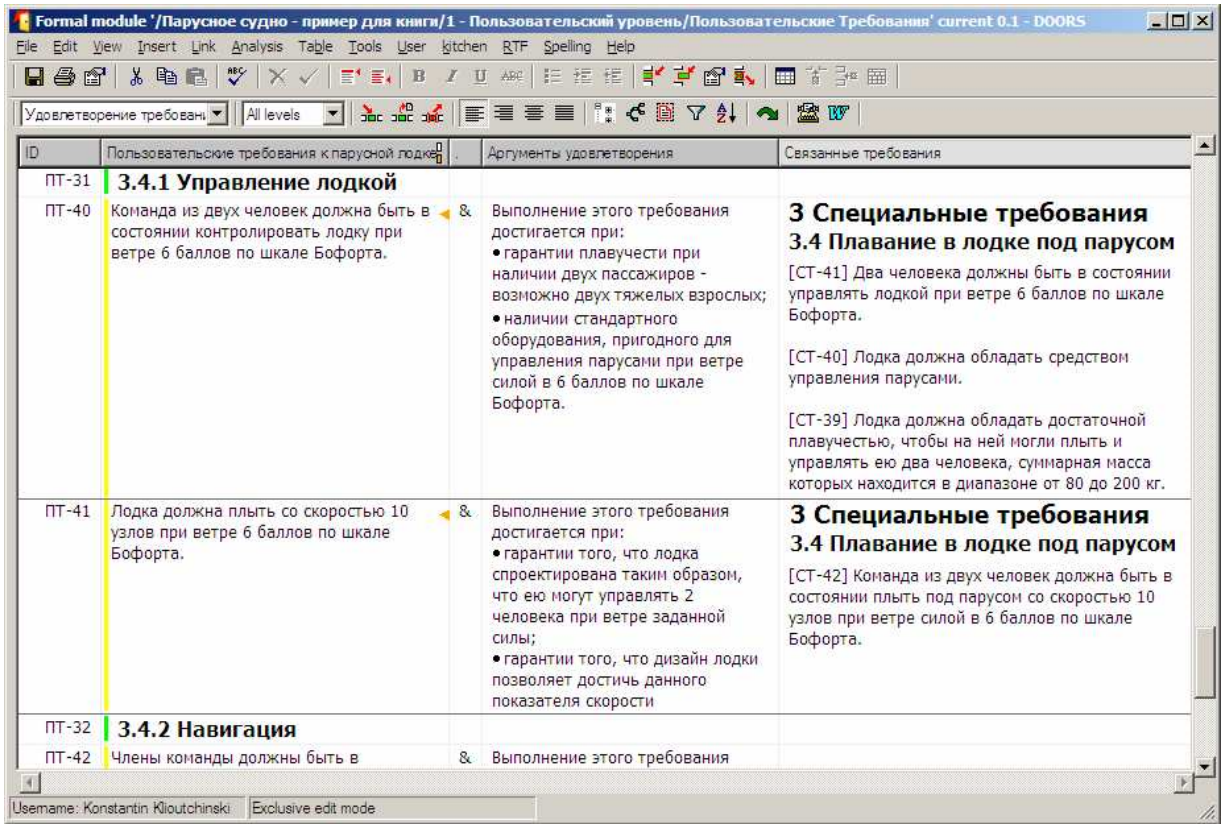


Рис. 9.13 Колонки трассировки, показывающие удовлетворяющие требования.

Данный вид существует в рамках модуля, содержащего пользовательские требования (текущий модуль), но в колонках трассировки отображены данные (требования и аргументы), которые находятся на другом конце входящих в модуль связей, т.е. хранятся в модуле с системными требованиями, которые призваны удовлетворить пользовательские требования.
В качестве примера использованы расширенные связи и в колонках вида (слева направо) представлены:

- идентификаторы пользовательских требований текущего модуля;
- основная колонка, показывающая заголовки/тексты пользовательских требований (из текущего модуля);
- оператор расширенной связи (атрибут пользовательского требования из текущего модуля);
- аргумент удовлетворения (атрибут пользовательского требования из текущего модуля);
- колонка «Связанные требования», которая показывает несколько атрибутов системных требований, связанных с пользовательским требованием. Вначале указывается идентификатор объекта системного требования, который показан жирным текстом и в квадратных скобках, а затем следует непосредственно текст системного требования. Дополнительно в этой же колонке показаны заголовки модуля системных требований, что позволяет лучше представить к какому именно разделу системных требований они относятся.

На рис. 9.14 показана колонка трассировки, но уже с другого конца тех же самых связей между системными и пользовательскими требованиями, т.е. уже со стороны системных требований.

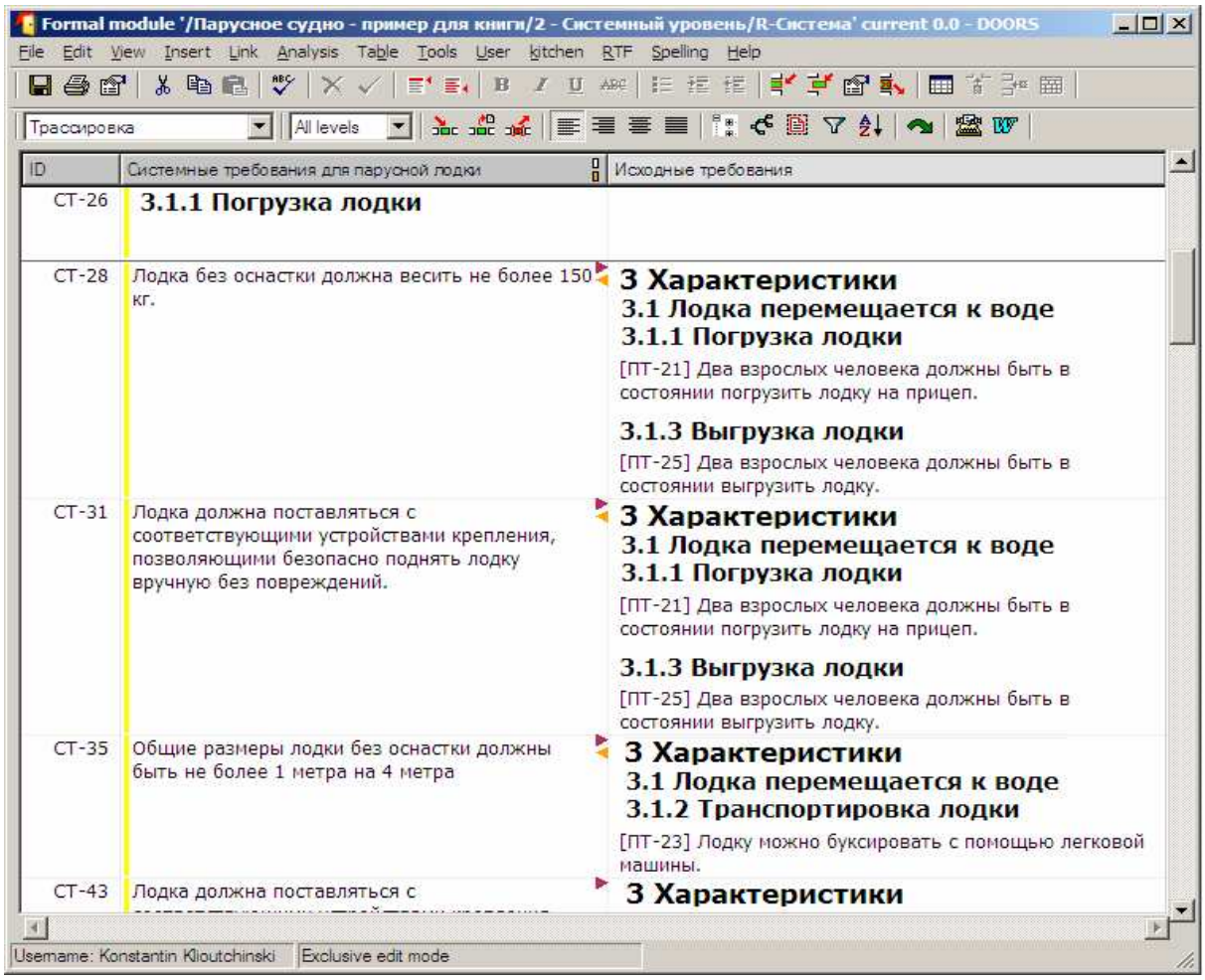


Рис. 9.14 Колонки трассировки для исходящих связей.

В этом случае показываются исходящие связи, которые ведут к информации, отображаемой в колонке с названием «Исходные требования» (колонка с аргументами удовлетворения отсутствует).

Весьма полезно при работе с документацией, содержащей требования, иметь под рукой матрицы трассировки (виды), демонстрирующие связи и отношения между документами (*напр.*, между требованиями разных уровней или между требованиями и приемочными тестами). Следует заметить, что в DOORS нет необходимости вручную обновлять информацию в таких матрицах, поскольку они будут обновляться автоматически.

9.8 Импорт и экспорт

Возможность обмена информацией между DOORS и другими приложениями является очень важным показателем. Эта функциональность нужна как для импорта имеющейся информации в DOORS, так и для экспорта данных из DOORS для последующего обмена с другими системами (*напр.*, с целью публикации).

Периодически в ходе проекта возникает необходимость быстро и без потерь импортировать и реорганизовать большой объем информации. Однако такие факторы, как большое количество различных источников информации, отличающиеся форматы и платформы хранения данных, несовместимость структур данных и т.д., могут сделать импорт очень сложной и трудоемкой задачей. DOORS предоставляет широкие возможности импорта и экспорта информации из документов, таблиц и баз данных.

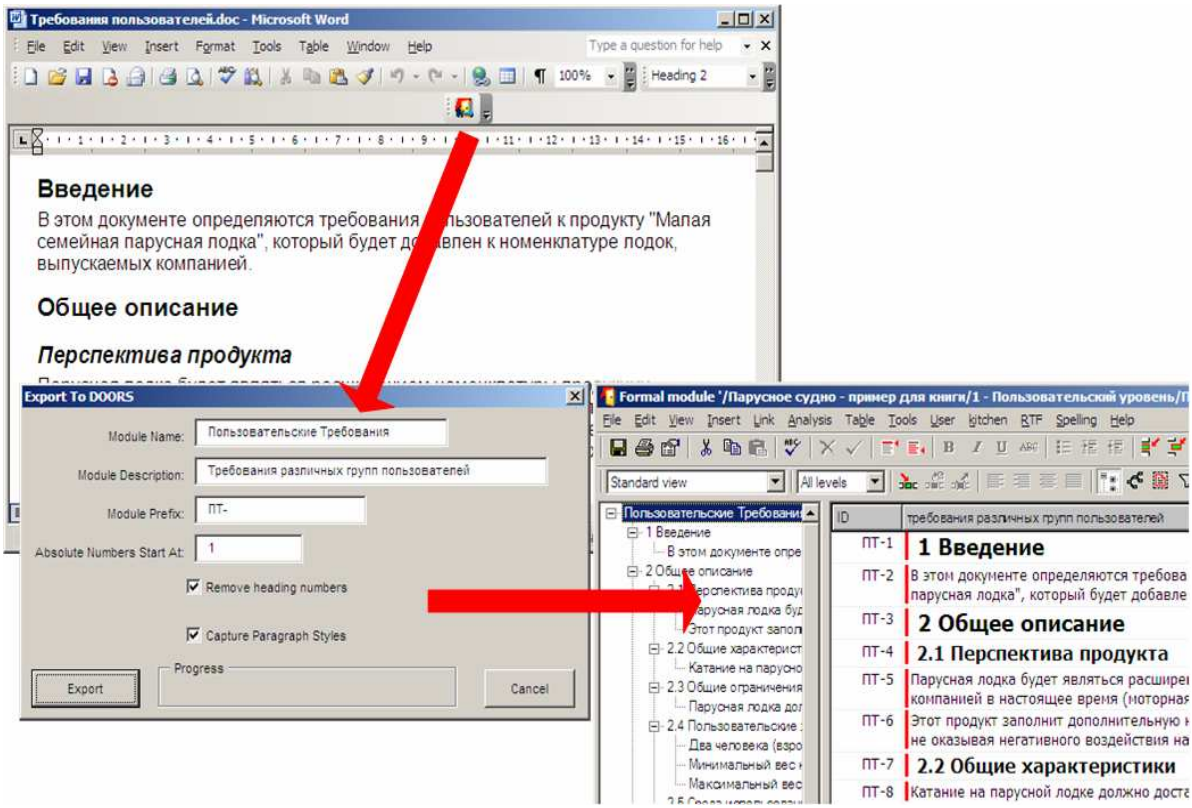


Рис. 9.15 Экспорт в DOORS из MS Word.

Так, например, на рис. 9.15 показана передача данных из документа MS Word в DOORS. Эту операцию можно легко выполнить с помощью специальной иконки на панели инструментов **Export to DOORS**, которая появляется в Word при инсталяции DOORS. В появившемся диалоговом окне следует лишь указать название модуля (возможно, нового), в который будут добавлены требования и внести его описание. При этом DOORS сохранит структуру исходного документа, т.е. Заголовок 1 (Heading 1) станет в DOORS объектом первого уровня (а границы объекта будет определять значок параграфа).

Аналогично, DOORS предоставляет возможность и экспорта в различные форматы, обеспечивая этим возможность передачи информации в ряд приложений. В качестве примера на рис. 9.16 показан экспорт данных из DOORS в Word.

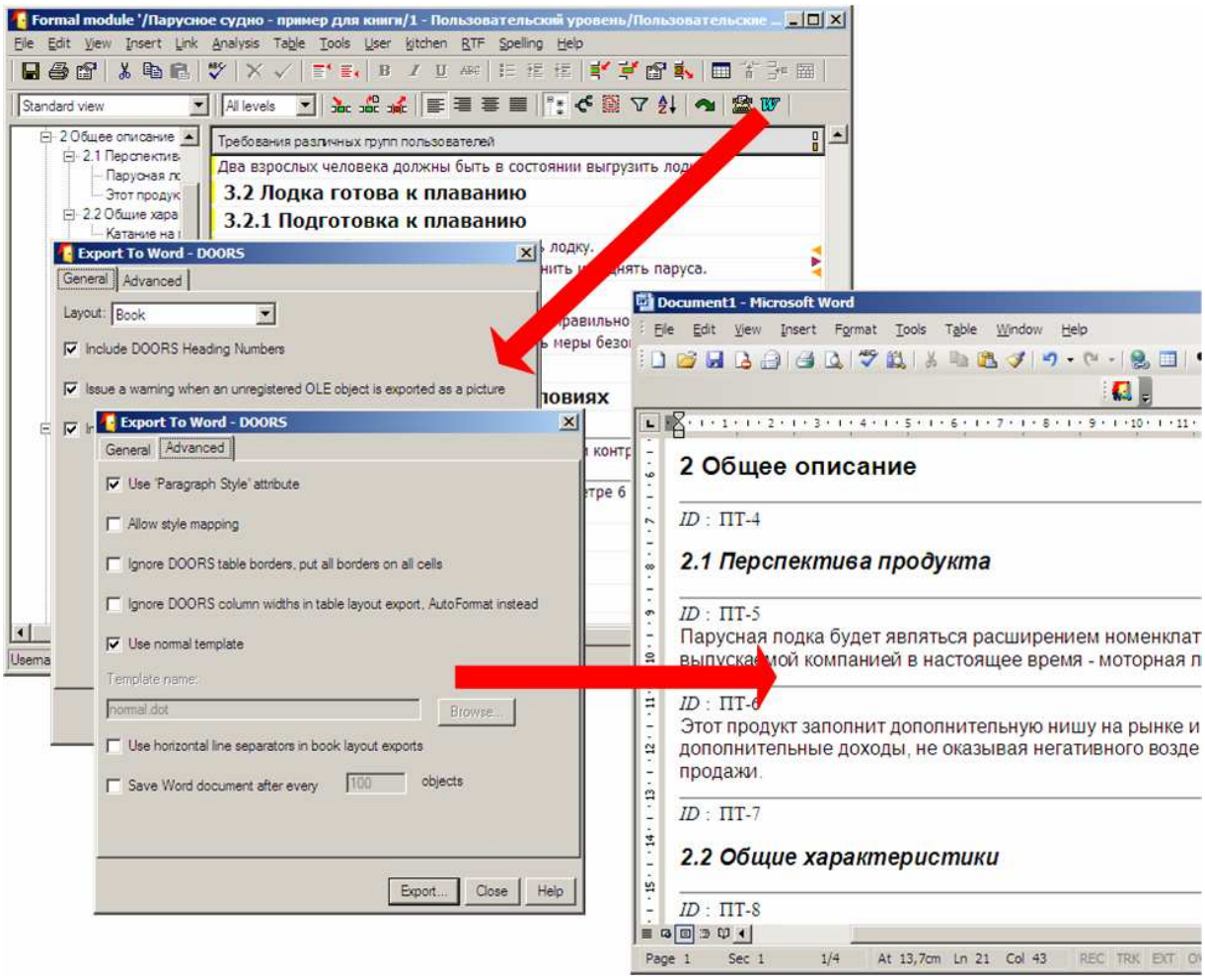


Рис. 9.16 Экспорт из DOORS в MS Word.

Эта операция является обратной предыдущей. При этом документ в Word сохранит ту же структуру, что и модуль в DOORS - заголовки первого уровня в иерархии объектов DOORS становятся заголовками первого уровня в Word и т.д., по уровням (в Word при выводе текста будет автоматически использоваться нормальный стиль - Normal style).

DOORS поддерживает функции экспорта и импорта для большого количества приложений и форматов, включая RTF, Word, WordPerfect, Excel, Lotus, Access, Plain Text, HTML, PowerPoint, MS Project, Outlook и многих других.

9.9 Моделирование на UML с помощью DOORS/Analyst

DOORS/Analyst это результат интеграции DOORS с продуктом Telelogic TAU G2 (системой моделирования на UML).
Таким образом, DOORS/Analyst позволяет создавать UML модели и диаграммы непосредственно в формальных модулях DOORS, размещая их внутри объектов.

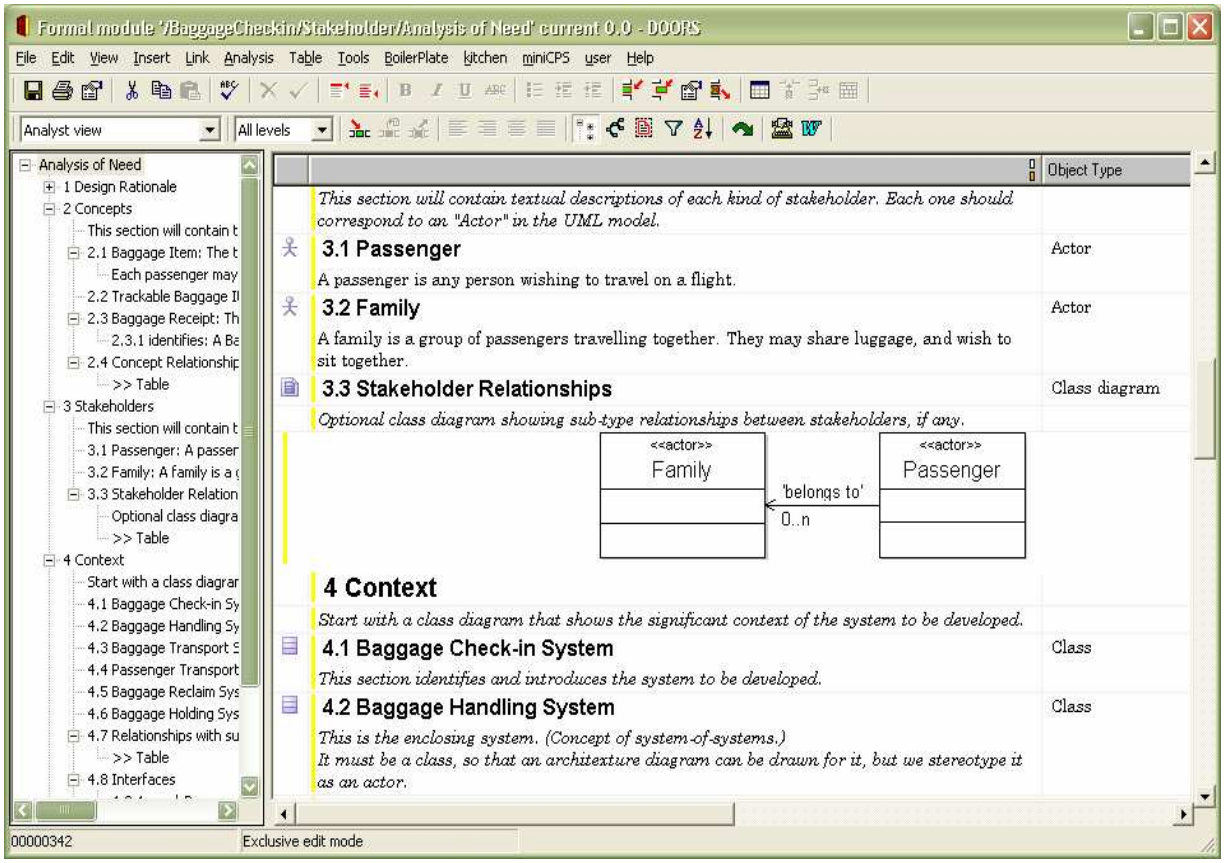


Рис. 9.17 Моделирование на UML с помощью DOORS/Analyst.

Более того, такие объекты могут быть помечены как элементы UML, например актеры, классы, состояния. И тогда, если какие-либо диаграммы с помощью опций главного меню вызываются и вставляются в модуль DOORS, то объекты, помеченные таким образом синхронизируются с элементами, которые появляются в диаграммах. Немаловажный результат такого подхода – возможность иметь трассировку требований вплоть до элементов внутри диаграммы.

На рис. 9.17 показан пример модуля DOORS, в котором для маркировки объектов в качестве элементов UML и вставки UML диаграммы классов использовался DOORS/Analyst. При этом объекты, маркированные как «UML element», не только помечаются специальными пиктограммами слева от основной колонки, но и справа от нее в столбце «Object Type» (тип объекта) указывается сущность UML элемента.

Двойное нажатие мышкой на UML диаграмме, запускает редактор диаграмм DOORS/Analyst, показанный на рис. 9.18.

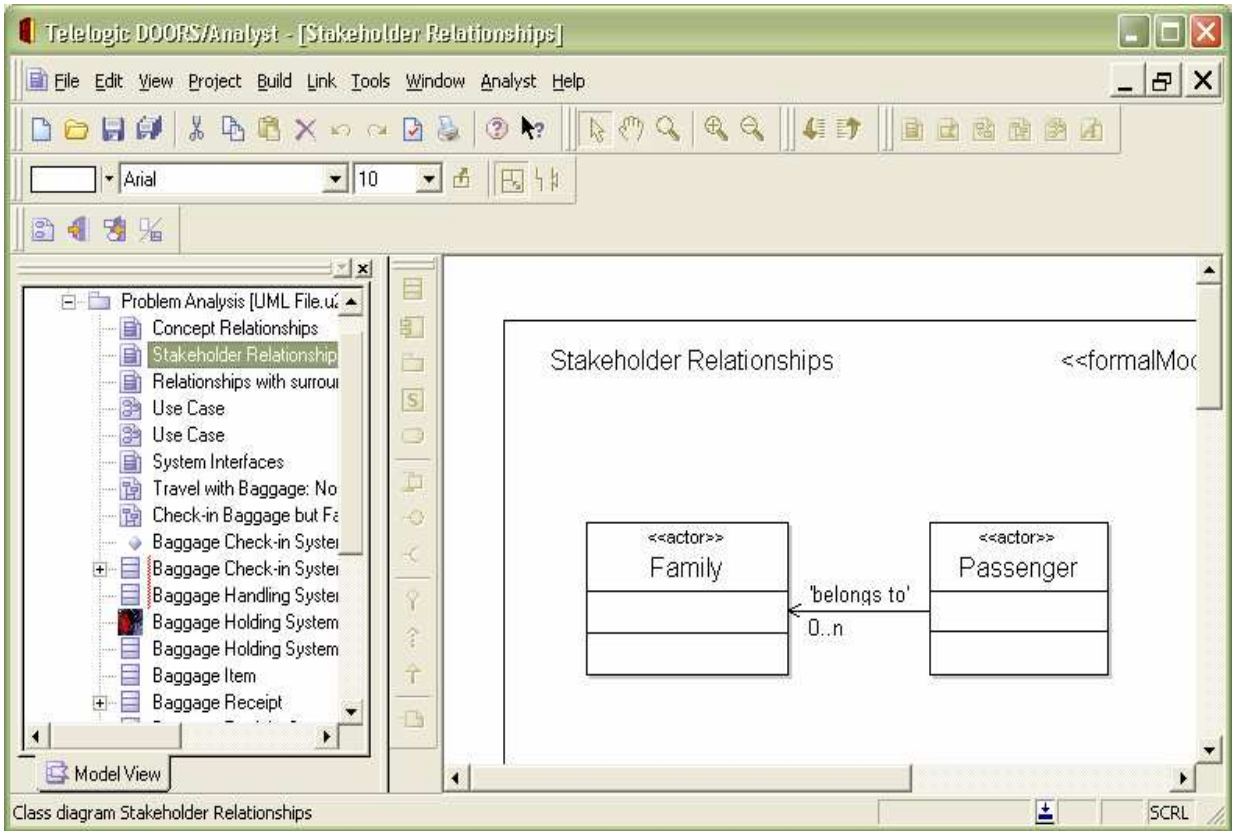


Рис. 9.18 Редактор UML диаграмм DOORS/Analyst.

При внесении каких-либо изменений в модель ее диаграммы автоматически синхронизируются с диаграммами формального модуля DOORS.

9.10 Заключение

В данном разделе мы дали краткое описание программного пакета DOORS, предназначенного для управления требованиями. Примеры, приводимые в данной главе, демонстрировали нам использование тех принципов, которые ранее излагались в этой книге: общий процесс разработки требований и метод расширенных связей.

В качестве примера того, как средство для моделирования может дополнить пакет для управления требованиями (или наоборот), был представлен инструмент DOORS/Analyst.

Все перечисленные в данной главе принципы и подходы могут быть также реализованы при помощи других средств управления требованиями.

Мы верим, что даже те, кто использует для разработки требований простой текстовый редактор, могут извлечь из этой книги много полезного для себя.

Библиография

Abrial J (1996) *The B Book.* Cambridge University Press.

Alderson A, Hull MEC, Jackson K and Griffiths LE (1998) Method Engineering for Industrial Real-time and Embedded Systems. *Information and Software Technology,* 40, 443—454.

Andriole SJ (1996) *Managing Systems Requirements: Methods, Tools and Cases.* New York, McGraw-Hffl.

Babich W (1986) *Software Configuration Management — Coordination for Team Productivity.* Addison-Wesley.

Bernstein P (1996) *Against the Gods — the Remarkable Story of Risk.* Wiley.

Bjorner D (1987) On the Use of Formal Methods in Software Development. *9th IEEE International Conference on Software Engineering, Washington, DC.*

Boehm B (1981) *Software Engineering Economics.* Prentice-Hall.

Booch G (1994) *Object-oriented Design with Applications.* Redwood City, CA, Benjamin Cummins.

Brown AW, Earl AN, et al. (1992) *Software Engineering Environments.* London, McGraw-Hill.

Budgen D (1994) *Software Design.* Addison-Wesley.

Chen P (1976) Entity—Relationship Approach to Data Modelling. *ACM Transactions on Database Systems,* 1(1), 9—36.

Clark KB and Fujimoto T (1991) *Product Development Performance.* Harvard Business School.

Coad P and Yourdon E (1991a) *Object-oriented Analysis.* Prentice-Hall.

Coad P and Yourdon E (1991 b) *Object-oriented Design.* Prentice-Hall.

Crosby PB (1979) *Quality is Free.* New York, McGraw-Hill.

Crosby PB (1984) *Quality without Tears.* New American Library.

Cooper RG (1993) *Winning at New Products.* Addison Wesley.

Darke P and Shanks G (1997) User Viewpoint and Modelling: Understanding and Representing User Viewpoints During Requirements Definition. *Information Systems Journal,* 7,213—239.

Davis AM (1993) *Software Requirements: Objects, Functions and States.* Englewood Cliffs, NJ, Prentice-Hall.

DeGrace P (1993) *The Olduvai Imperative: CASE and the State of Software Engineering Practice.* Prentice-Hall International.

DeMarco T (1978) *Structured Analysis and System Specification.* Yourdon Press.

DeMarco T (1982) *Controlling Software Projects.* Yourdon Press.

DeMarco T and Lister T (1987) *Peopleware — Productive Projects and Teams.* Dorset House.

Easterbrook S and Nuseibeh B (1996) Using Viewpoints for Inconsistency Management. *Software Engineering Journal,* 11(1), 31—43.

Finkeistein A, Kramer J, Nuseibeh B and Goedicke M (1992) Viewpoints: A Framework for Integrating Multiple Perspectives in Systems Development. *International Journal of Software Engineering and Knowledge Engineering,* 2(10), 3 1—58.

Fowler M and Scott K (1997) *UML Distilled: Applying the Standard Object Modeling Language.* Reading, MA, Addison-Wesley.

Gilb T (1988) *Principles of Software Engineering Management.* Addison-Wesley.

Gorchels L (1997) *The Product Manager's Handbook.* NTC Business Books.

Gotel OCZ and Finkelstein ACW (1995) Contribution Structures. In *Proceedings RE '95,* York, IEEE Press.

Harel D (1987) Statecharts: a Visual Formalism for Complex Systems. *Science of Computer Programming,* 8, 231—274.

Hull MEC, Taylor PS, Hanna JRP and Millar RJ (2002) Software Development Processes —An Assessment. *Information and Software Technology,* 44(1), 1—12.

Humphrey WM (1989) *Managing the Software Process.* Addison-Wesley.

Jackson M (1995) *Software Requirements and Specifications: a Lexicon of Practice, Principles and Prejudices.* Addison-Wesley.

Jacobsen I, Christerson M, et al. (1993) *Object-oriented Software Engineering.* Wokingham, MA, Addison-Wesley.

Jones CB (1986) *Systematic Software Development Using VDM*. London, Prentice-Hall.

Kotonya G and Sommerville I (1996) Requirements Engineering with Viewpoints. *Software Engineering Journal,* 11(1), 5—11.

Kotonya G and Sommerville I (1998) *Requirements Engineering: Processes and Techniques.* Chichester, Wiley.

Leite JCP and Freeman PA (1991) Requirements Validation through Viewpoint Resolution. *Transactions of Software Engineering,* 12(2), 1253—1269.

Loucopulos P and Karakostas V (1995) *Systems Requirements Engineering.* McGraw-Hill.

Mazza C, et al. (1994) *ESA — Software Engineering Standards.* Prentice-Hall.

Monroe RT, Kompanek A, Metlon R and Garlan D (1997) *Architectural Styles, Design Patterns, and Objects.* IEEE Software.

Mumford E (1989) User Participation in a Changing Environment —Why We Need IT. In *Participation in Systems Development,* ed. Knight K. London, Kogan Page.

Nuseibeh B, Kramer I and FinkelsteinA (1994) A Framework for Expressing the Relationships between Multiple Views in Requirements Specification. *Transactions of Software Engineering,* 20(10), 760—773.

Oliver DW, Kelliher TP and Keegan JG (1997) *Engineering Complex Systems with Models and Objects.* New York, McGraw- Hill.

OMG (2003) *The Unified Modelling Language Version 2*, www.omg.org.

Page-Jones M (1980) *The Practical Guide to Structured Systems.* Yourdon Press.

Perrow C (1984) *Normal Accidents.* Basic Books.

Petroski H (1982) *To Engineer is Human — the Role of Failure in Successful Design.* St Martin's Press.

Petroski H (1996) *Invention by Design: How Engineers Get from Thought to Thing.* Harvard University Press.

Poots C, Takahashi K, et al. (1994) Inquiry-based Requirements Analysis. *IEEE Software,* 11(2), 2 1—32.

Pressman RS (1997) *Software Engineering: a Practitioner's Approach.* McGraw-Hill.

Ross DT (1977) Structured Analysis (SA): a Language for Communicating Ideas. *IEEE Transactions on Software Engineering,* 3(1), 16—34.

Ross DT (1985) Applications and Extensions of SADT. *IEEE Computer,* 18(4), 25—34.

Ross DT and Schoman KE (1977) Structured Analysis for Requirements Definition. *IEEE Transactions on Software Engineering,* 3(1), 6—15.

Rumbaugh J, Blaha M, et al. (1991a) *Object Modeling and Design.* Englewood Cliffs, NJ, Prentice-Hall.

Rumbaugh J, Blaha M, Premerlani W, Eddy F and Lorenzen W (1991b) *Object- oriented Modeling and Design.* Prentice-Hall.

Shlaer S and Mellor SI (1991) *Object Life Cycles — Modeling the World in States.* Yourdon Press.

Shlaer S and Mellor SI (1998) *Object-oriented Systems Analysis.* Englewood Cliffs, NJ, Prentice-Hall.

Software Engineering Institute (Carnegie-Mellon) (1991) *Capability Maturity Model for Software. Tech. Report CMU/SEI-91 - TR-24.*

Sommervile I (1996) *Software Engineering.* Wokingham, MA, Addison-Wesley. Sommervile I and Sawyer P (1997) *Requirements Engineering: a Good Practice Guide.* Chichester, Wiley.

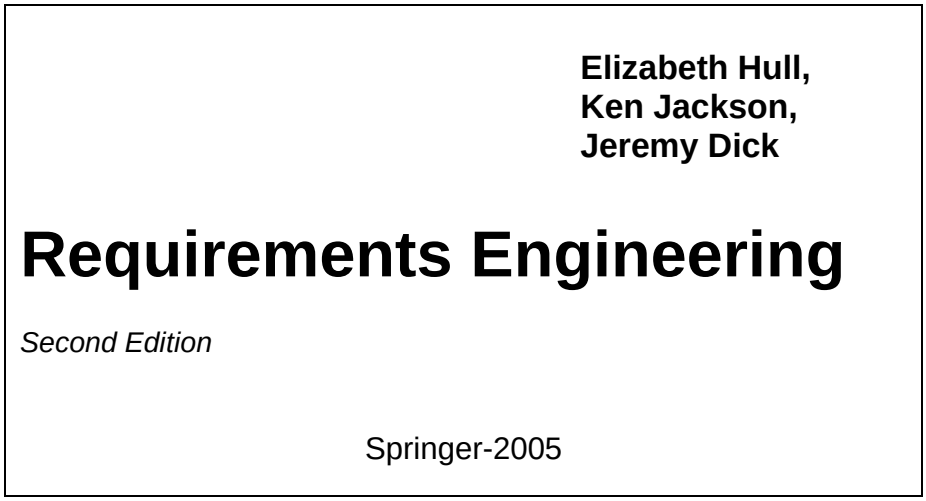
SpiveyJM (1989) *The ZNotation: a Reference Manual.* Prentice-Hall.

Stevens R, Brook P, Jackson K and Arnold S (1998) *Systems Engineering: Coping with Complexity.* Prentice-Hall Europe.

Yourdon EN (1990) *Modern Structured Analysis.* Prentice-Hall.

Дополнительная информация

Заказать оригинал книги или получить дополнительную информацию о ней, вы сможете посетив сайт:
http://www.requirementsengineering.info



Для получения информации о компании Telelogic и ее инструментах:

Enterprise System Architect
Focal Point
Doors
Doors\Analyst
Doors XT
Synergy\Change
Synergy\CM
Tau G2
Tau Tester
Tau SDL
Tau TTCN
Logiscope
DocExpress

вы можете посетить сайт компании: www.telelogic.com
или обратиться с запросом по адресу: anatoly.volokhov@telelogic.com