РАЗРАБОТКА МОДЕЛЕЙ СЛОЖНЫХ СИСТЕМ НА ЯЗЫКЕ UML

Москва
2019

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Московский государственный
гуманитарно-экономический университет

Разработка моделей сложных системна языке UML

Учебно-методическое пособие

Москва
2019

УДК 681.3.06
ББК 32.973.018
И 89

Рецензенты:

А.А. Малашин, доктор физико-математических наук, профессор, заведующий кафедрой «Прикладная математика, информатика и вычислительная техника» ФГБОУ ВПО МФ МГТУ им. Н.Э. Баумана.

А.Е. Никольский, кандидат технических наук, доцент кафедры «Прикладная математика и информатика по областям» ФГБОУИ ВО МГГЭУ.

Истомина Т.В., Петрунина Е.В., Белоглазов А.А., Байрамов Э.В.

И 89 Разработка моделей сложных систем на языке UML: *учебно-методическое пособие*. – М.: МГГЭУ, 2019. – 104 с.

В учебно-методическом пособии рассмотрены теоретические и практические основы проектирования на языке UML. Описаны базовые принципы унифицированного процесса разработки моделей сложных систем с использованием языка UML, основные этапы разработки, рабочие процессы и создаваемые артефакты. В пособии также даны примеры решения типовых задач и задания для выполнения практических работ по рассматриваемым темам.

Для студентов бакалавриата, обучающихся по направлениям подготовки «Прикладная математика и информатика», «Прикладная информатика» и «Информатика и вычислительная техника», изучающих дисциплины «Проектирование информационных систем», «Технологии программирования» «Проектный практикум», «Медицинские информационные системы», «Биотехнические системы и технологии». Пособие может использоваться в процессе подготовки выпускной квалификационной работы бакалавра.

Для студентов магистратуры, обучающихся по направлениям подготовки «Прикладная математика и информатика», «Прикладная информатика», изучающих дисциплины «Методы и модели системного анализа», «Современные методы и средства разработки программного обеспечения», «Методология и технология проектирования информационных систем», «Системы поддержки принятия решений врача», «Методы и модели системного анализа биосистем». Пособие может использоваться в процессе подготовки выпускной квалификационной работы магистра.

ISBN 978-5-9799-0128-2

Краткое содержание

Предисловие 6
Введение 8

1. Теоретические сведения о разработке программного обеспечения на языке UML 10
2. Анализ требований к программному продукту 20
3. Основы программирования на языке UML 43
4. Основные сведения о работе в среде Rational Rose 54
5. Задания к практическим работам 66

Практическая работа № 1. Составление технического задания на разработку ИС 67
Практическая работа № 2. Объектно-ориентированный анализ предметной области 68
Практическая работа № 3. Разработка требований к системе 76
Практическая работа № 4. Визуализация бизнес-процессов 83

Приложения

Приложение 1. Приемы формирования требований 89
Приложение 2. Пример разработанного технического задания на проектирование ИС 92
Техническое задание 93

Литература 100

Полное оглавление

Предисловие 6
Введение 8

1. Теоретические сведения о разработке программного обеспечения на языке UML 10
   1.1. Общие сведения о разработке программного обеспечения 10
   1.2. Процесс управления разработкой программного обеспечения 12
   1.3. Планирование проекта разработки программного обеспечения 15
   1.4. Общие сведения о требованиях к информационным системам 17
2. Анализ требований к программному продукту 20
   2.1. Виды требований 20
   2.2. Взаимодействие участников в процессе разработки требований 25
   2.3. Процесс разработки требований 29
   2.4. Разработка технического задания на проектирование информационных систем 37
3. Основы программирования на языке UML 43
   3.1. Назначение и описание языка UML 43
   3.2. Способы использования UML 47
   3.3. Модель UML и ее элементы 47
4. Основные сведения о работе в среде Rational Rose 54
5. Задания к практическим работам 66
Практическая работа № 1. Составление технического задания на разработку ИС 67
Практическая работа № 2. Объектно-ориентированный анализ предметной области 68
Практическая работа № 3. Разработка требований к системе 76

Практическая работа № 4. Визуализация бизнес-процессов 83
Приложения
Приложение 1. Приемы формирования требований 89
Приложение 2. Пример разработанного технического задания на проектирование ИС 92
Техническое задание 93

Литература 100

ПРЕДИСЛОВИЕ

Учебно-методическое пособие содержит теоретические и практические основы проектирования на языке UML. В данном учебном пособии рассмотрены базовые принципы унифицированного процесса разработки программного обеспечения с использованием языка UML, основные этапы разработки (включая сбор и формализацию требований, анализ и проектирование, реализацию и тестирование), а также рабочие процессы и создаваемые артефакты.

Учебное пособие предназначено для студентов, изучающих дисциплины «Проектирование информационных систем», «Программная инженерия», «Проектный практикум», «Биотехнические системы и технологии», а также другие дисциплины, посвященные вопросам разработки программного обеспечения информационных систем.

Для изучения учебного пособия необходимо наличие у студентов базовых знаний в области объектно-ориентированного программирования и в области баз данных.

Структура пособия построена в соответствии с последовательностью практических и лабораторных работ по дисциплинам «Проектирование информационных систем», «Программная инженерия». Цели и задачи соответствуют программам указанных дисциплин и способствуют формированию соответствующих компетенций основной образовательной программы.

В ходе освоения дисциплины студент должен *знать*:

- основные понятия, приемы и методы анализа требований к проекту и техническому заданию на разработку автоматизированной информационной системы;
- основные понятия, приемы и методы разработки архитектуры системы;
- основные методы анализа и синтеза сложных проектных решений разработки информационных систем;

уметь:

- осуществлять постановку задачи анализа и синтеза сложных проектных решений;
- осуществлять распределение заданий и ресурсов в соответствии с жизненным циклом программно-аппаратных систем;
- разрабатывать программное обеспечение с использованием языка UML;
- использовать современные информационные технологии и инструментальные средства для решения задач проектирования на языке UML;

владеть навыками:

- применения современных программных средств для разработки проектно-конструкторской и технологической документации;
- применения методов и способов оценки архитектурных решений и проектов на соответствие требованиям технического задания и стандартам;
- проектирования программных компонентов с учетом требований, разработанных интерфейсов и с использованием современных инструментальных средств.

ВВЕДЕНИЕ

Современный этап развития информационных технологий промышленной и индивидуальной разработки программного обеспечения характеризуется развитием и внедрением объектно-ориентированного подхода, основанного на применении различных пакетов визуальной разработки, таких как Java DSK, Microsoft Visual Studio, Qt SDK и другие. Данные средства разработки поставили в основу процесса разработки информационных систем этапы анализа и проектирования.

Несмотря на более чем пятидесятилетнее существование компьютерной отрасли, многие компании — разработчики программного обеспечения главными своими задачами считают процессы документирования и управления требованиями к программному обеспечению [13]. Основными факторами, определяющими векторы современного развития процесса разработки ПО, являются:

- признание бизнес-анализа профессиональной деятельностью;
- высокий уровень развития средств управления требованиями в базах данных, а также поддержки разработки требований;
- всестороннее использование методов гибкой разработки (agile) и развитие приемов работы с требованиями в проектах гибкой разработки;
- активное использование визуальных моделей для представления знаний о требованиях.

Таким образом, основной задачей разработчика программных систем является процесс формализации исходных требований, дальнейший анализ и проектирование, результаты которого транслируются в исходные коды программ. Автоматизация процесса анализа требований осуществляется посредством различных программных CASE средств (Computer-Aided Software Engineering), направленных на разработку и преобразование моделей программного проекта. CASE-технология представляет собой совокупность методологий анализа, проектирования, разработки и сопровождения сложных систем программного обеспечения, поддержанную комплексом взаимоувязанных средств автоматизации. Разработка как крупных, так и небольших проектов информационных систем возможна при помощи CASE-средств различных производителей, в том числе с открытым кодом, с частичной или полной автоматизацией проектирования.

В настоящее время существует два основных подхода к проектированию:

- функционально-ориентированный (структурный) метод;
- объектно-ориентированный метод.

В данном учебно-методическом пособии рассматривается объектно-ориентированный подход к разработке ПО.

1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ О РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ЯЗЫКЕ UML

1.1. Общие сведения о разработке программного обеспечения

Разработка ПО является бурно развивающимся процессом, последние десятилетия подтверждают данный факт. Проблемы управления процессами разработки программного обеспечения проявились в 60-х – 70-х годах, когда в течение нескольких лет потерпел неудачу целый ряд крупных проектов по разработке программных продуктов. В этот период многие разработчики ПО столкнулись с проблемами задержек в разработке, многие проекты оказались ненадежными, а затраты на разработку в несколько раз превосходили первоначальные оценки, созданные программные системы часто имели низкие показатели производительности. Основной причиной неудач при разработке ПО послужил принятый тогда подход к процессу его разработки. Методика разработки программного обеспечения того времени делала упор на методы и приемы реализации технической стороны проектов и не уделяла должного внимания анализу бизнес-целей проектов, что и стало причиной неэффективности разработки программного обеспечения информационных систем.

Необходимость управления программными проектами вытекает из того факта, что процесс создания профессионального ПО всегда является объектом бюджетной политики организации, где оно раз-

рабатывается, и имеет временные ограничения. Работа руководителя программного проекта по большому счету заключается в том, чтобы гарантировать выполнение этих бюджетных и временных ограничений с учетом бизнес-целей организации относительно разрабатываемого ПО.

Руководители проектов призваны спланировать все этапы разработки программного продукта. Они также должны контролировать ход выполнения работ и соблюдения всех требуемых стандартов. Постоянный контроль за ходом выполнения работ необходим для того, чтобы процесс разработки не выходил за временные и бюджетные ограничения. Хорошее управление не гарантирует успешного завершения проекта, но плохое управление обязательно приведет к его провалу. Это может выразиться в задержке сроков сдачи готового ПО, в превышении сметной стоимости проекта и в несоответствии готового ПО спецификации требований.

Рассмотрим основные особенности процессов разработки ПО, которые требуют использования специальных методов и средств разработки.

1. Нематериальность программного обеспечения, которая проявляется в невозможности физически отследить данный процесс. Поскольку руководитель программного проекта не может видеть весь процесс развития разрабатываемого ПО, основным источником информации о ходе реализации проекта будет документация, которая фиксирует процесс разработки программного продукта.
2. Разработка крупных информационных систем, как правило, требует индивидуального подхода и сильно отличается от проектов, реализующих небольшое ПО. Таким образом, полученные навыки при разработке небольших проектов не всегда приемлемы, постоянные технологические изменения в компьютерной технике и коммуникационном оборудовании могут обесценить предыдущий опыт.

Перечисленные особенности могут привести к тому, что реализация проекта выйдет из временного графика или превысит бюджет-

ные ассигнования. Программные системы зачастую оказываются новинками, как в идеологическом, так и в техническом плане. Таким образом, проблемы в реализации программного проекта могут привести к изменению бюджетных и временных рамок проектов.

1.2. Процесс управления разработкой программного обеспечения

Процесс разработки ПО осуществляется в соответствии с нормативными документами и стандартами в области информационных технологий [1–2]. Данный процесс индивидуален для каждого проекта в зависимости от организации, типа создаваемого программного продукта. Однако необходимо отметить основные этапы работ:

- формулирование предложений по созданию ПО;
- планирование и составление графика работ по созданию ПО;
- оценивание стоимости проекта;
- подбор персонала;
- контроль хода выполнения работ;
- написание отчетов и представлений.

Предложения по созданию ПО, формулируемые на первом этапе разработки, должны содержать описание целей проектов и способов их достижения. Они также обычно включают в себя оценки финансовых и временных затрат на выполнение проекта. При необходимости здесь могут приводиться обоснования для передачи проекта на выполнение сторонней организации или команде разработчиков.

Написание предложений — очень ответственная работа, так как для многих организаций вопрос о том, будет ли проект выполняться самой организацией или разрабатываться по контракту сторонней компанией, является критическим. Не существует каких-либо рекомендаций по написанию предложений, многое здесь зависит от опыта.

На этапе планирования проекта определяются процессы, этапы и ожидаемые на каждом из них результаты, которые должны привести

к выполнению проекта. Реализация этого плана должна привести к достижению целей проекта.

Определение стоимости проекта напрямую связано с его планированием, поскольку здесь оцениваются ресурсы, требующиеся для выполнения плана.

Мониторинг проекта (контроль хода выполнения работ) — это непрерывный процесс, продолжающийся в течение всего срока реализации проекта. Руководитель должен постоянно отслеживать ход реализации проекта и сравнивать фактические и плановые показатели выполнения работ с их стоимостью. Хотя многие организации имеют механизмы формального мониторинга работ, опытный руководитель может составить ясную картину о стадии развитии проекта просто путем неформального общения с разработчиками.

Неформальный мониторинг часто помогает обнаружить потенциальные проблемы, которые в явном виде могут обнаружиться позднее. Например, ежедневное обсуждение хода выполнения работ может выявить отдельные недоработки в создаваемом программном продукте. Вместо ожидания отчетов, в которых будет отражен факт «пробуксовки» графика работ, можно обсудить со специалистами намечающиеся программистские проблемы и не допустить срыва графика работ.

В ходе реализации проекта обычно происходит несколько формальных контрольных проверок хода выполнения работ по созданию ПО. Такие проверки должны дать общую картину хода реализации проекта в целом и показать, насколько уже разработанная часть ПО соответствует целям проекта.

Время выполнения больших программных проектов может занимать длительный период. В течение этого времени цели и намерения организации, заказавшей программный проект, могут существенно измениться, разрабатываемый программный продукт может стать ненужным, исходные требования к создаваемому ПО — измениться. В такой ситуации руководство организации-разработчика может принять решение о прекращении разработки ПО или об изменении проекта в целом с тем, чтобы учесть изменившиеся цели и намерения организации-заказчика.

Руководители проектов обычно обязаны сами подбирать исполнителей для своих проектов. В идеальном случае профессиональный уровень исполнителей должен соответствовать той работе, которую они будут выполнять в ходе реализации проекта. Однако во многих случаях руководители должны полагаться на команду разработчиков, которая далека от идеальной. Такая ситуация может быть вызвана следующими причинами.

1. Бюджет проекта не позволяет привлечь высококвалифицированный персонал. В таком случае за меньшую плату привлекаются менее квалифицированные специалисты.
2. Существуют ситуации, когда невозможно найти специалистов необходимой квалификации как в самой организации-разработчике, так и вне ее.
3. Организация хочет повысить профессиональный уровень своих работников. В этом случае она может привлечь к участию в проекте неопытных или недостаточно квалифицированных работников, чтобы они приобрели необходимый опыт и поучились у более опытных специалистов.

Таким образом, почти всегда подбор специалистов для выполнения проекта имеет определенные ограничения и не является свободным. Вместе с тем необходимо, чтобы хотя бы несколько членов группы разработчиков имели квалификацию и опыт, достаточные для работы над данным проектом. В противном случае сложно избежать непоправимых ошибок в разработке ПО.

Руководитель проекта обычно обязан посылать отчеты о ходе его выполнения как заказчику, так и подрядным организациям. Это должны быть краткие документы, основанные на информации, извлекаемой из подробных отчетов о проекте. В этих отчетах должна быть та информация, которая позволяет четко оценить степень готовности создаваемого программного продукта.

Можно выделить следующие роли в группе по разработке ПО:

- руководитель — общее руководство проектом, написание документации, общение с заказчиком ПО;

- системный аналитик — разработка требований (составление технического задания, проекта программного обеспечения);
- тестер — составление плана тестирования и аттестации готового ПО (продукта), составление сценария тестирования, базовый пример, проведение мероприятий по плану тестирования;
- разработчик — моделирование компонентов программного обеспечения, кодирование.

1.3. Планирование проекта разработки программного обеспечения

Эффективное управление программным проектом напрямую зависит от правильного планирования работ, необходимых для его выполнения. План помогает руководителю предвидеть проблемы, которые могут возникнуть на каких-либо этапах создания ПО, и разработать превентивные меры для их предупреждения или решения. План, разработанный на начальном этапе проекта, рассматривается всеми его участниками как руководящий документ, выполнение которого должно привести к успешному завершению проекта. Этот первоначальный план должен максимально подробно описывать все этапы реализации проекта.

Процесс планирования начинается, исходя из описания системы, с определения проектных ограничений (временные ограничения, возможности наличного персонала, бюджетные ограничения и т.д.). Эти ограничения должны определяться параллельно с оцениванием проектных параметров, таких как структура и размер проекта, а также распределение функций среди исполнителей. Затем определяются этапы разработки и то, какие результаты (документация, прототипы, подсистемы или версии программного продукта) должны быть получены по окончании этих этапов. Далее начинается циклическая часть планирования. Сначала разрабатывается график работ по выполнению проекта или дается разрешение на продолжение использования ранее созданного графика. После этого проводится

контроль выполнения работ и отмечаются расхождения между реальным и плановым ходом работ.

Далее, по мере поступления новой информации о ходе выполнения проекта, возможен пересмотр первоначальных оценок параметров проекта. Это, в свою очередь, может привести к изменению графика работ. Если в результате этих изменений нарушаются сроки завершения проекта, должны быть пересмотрены и согласованы с заказчиком ПО проектные ограничения.

Конечно, большинство руководителей проектов не думают, что реализация их проектов пройдет гладко, без всяких проблем. Желательно описать возможные проблемы еще до того, как они проявят себя в ходе выполнения проекта. Поэтому лучше составлять пессимистические графики работ, чем оптимистические. Но, конечно, невозможно построить план, учитывающий все, в том числе случайные, проблемы и задержки выполнения проекта, поэтому и возникает необходимость периодического пересмотра проектных ограничений и этапов создания программного продукта.

План проекта должен четко показать ресурсы, необходимые для реализации проекта, разделение работ на этапы и временной график выполнения этих этапов. В некоторых организациях план проекта составляется как единый документ, содержащий все виды планов, описанных выше. В других случаях план проекта описывает только технологический процесс создания ПО. В таком плане обязательно присутствуют ссылки на планы других видов, но они разрабатываются отдельно от плана проекта.

Детализация планов проектов очень различается в зависимости от типа разрабатываемого программного продукта и организации-разработчика. Но в любом случае большинство планов содержат следующие разделы.

1. Введение — краткое описание целей проекта и проектных ограничений (бюджетных, временных и т.д.), важных для управления проектом.
2. Организация выполнения проекта — описание способа подбора команды разработчиков и распределение обязанностей между ними.

3. Анализ рисков. Описание возможных проектных рисков, вероятности их проявления и стратегий, направленных на их уменьшение.
4. Перечень аппаратных средств и программного обеспечения, необходимого для разработки программного продукта. Если аппаратные средства требуется закупать, приводится их стоимость совместно с графиком закупки и поставки.
5. Разбиение работ на этапы. Процесс реализации проекта разбивается на отдельные процессы, определяются этапы выполнения проекта, приводится описание результатов (выходов) каждого этапа и контрольные отметки.
6. График работ. В этом графике отображаются зависимости между отдельными процессами (этапами) разработки ПО, оценки времени их выполнения и распределение членов команды разработчиков по отдельным этапам.
7. Механизмы мониторинга и контроля хода выполнения проекта. Описываются предоставляемые руководителем отчеты о ходе выполнения работ, сроки их предоставления, а также механизмы мониторинга всего проекта.

План должен регулярно пересматриваться в процессе реализации проекта. Одни части плана, например график работ, изменяются часто, другие более стабильны. Для внесения изменений в план требуется специальная организация документопотока, позволяющая отслеживать эти изменения.

1.4. Общие сведения о требованиях к информационным системам

Проблемы, которые приходится решать специалистам в процессе создания программного обеспечения, очень сложны. Природа этих проблем не всегда ясна, особенно если разрабатываемая программная система инновационная. В частности, трудно чётко описать те действия, которые должна выполнять система. Описание функциональных возможностей и ограничений, накладываемых на систему,

называется требованиями к этой системе, а сам процесс формирования, анализа, документирования и проверки этих функциональных возможностей и ограничений — разработкой требований.

Требования подразделяются на пользовательские и системные. Пользовательские требования — это описание на естественном языке (плюс поясняющие диаграммы) функций, выполняемых системой, и ограничений, накладываемых на неё. Системные требования — это описание особенностей системы (архитектура системы, требования к параметрам оборудования и т.д.), необходимых для эффективной реализации требований пользователя.

Разработка требований — это процесс, включающий мероприятия, необходимые для создания и утверждения документа, содержащего спецификацию системных требований. Для новых программных систем процесс разработки требований должен начинаться с анализа осуществимости. Началом такого анализа является общее описание системы и ее назначения, а результатом анализа — отчет, в котором должна быть четкая рекомендация, продолжать или нет процесс разработки требований проектируемой системы. Другими словами, анализ осуществимости должен осветить следующие вопросы.

1. Отвечает ли система общим и бизнес-целям организации-заказчика и организации-разработчика?
2. Можно ли реализовать систему, используя существующие на данный момент технологии и не выходя за пределы заданной стоимости?
3. Можно ли объединить систему с другими системами, которые уже эксплуатируются?

Критическим является вопрос, будет ли система соответствовать целям организации. Если система не соответствует этим целям, она не представляет никакой ценности для организации.

Выполнение анализа осуществимости включает сбор и анализ информации о будущей системе и написание соответствующего отчета. Сначала следует определить, какая именно информация необходима, чтобы ответить на поставленные выше вопросы. На-

пример, эту информацию можно получить, ответив на следующее вопросы.

1. Что произойдет с организацией, если система не будет введена в эксплуатацию?
2. Какие текущие проблемы существуют в организации и как новая система поможет их решить?
3. Каким образом система будет способствовать целям бизнеса?
4. Требует ли разработка системы технологии, которая до этого не использовалась в организации?

Далее необходимо определить источники информации. Это могут быть менеджеры отделов, где система будет использоваться, разработчики программного обеспечения, знакомые с типом будущей системы, технологи, конечные пользователи и т.д.

После обработки собранной информации готовится отчет по анализу осуществимости создания системы. В нем должны быть даны рекомендации относительно продолжения разработки системы. Могут быть предложены изменения бюджета и графика работ по созданию системы или предъявлены более высокие требования к системе.

2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ПРОДУКТУ

2.1. Виды требований

Все требования к ПО можно подразделить на три уровня: бизнес-требования. пользовательские требования и функциональные требования. Краткая информация о требованиях приведена в табл. 1, а их взаимодействие показано на рис. 2.1 [13].

Таблица 1

Информация о некоторых типах требований

Понятие	Определение
Бизнес-требование	Высокоуровневая бизнес-цель организации или заказчиков системы.
Бизнес-правило	Политика, предписание, стандарт или правило, определяющее или ограничивающее некоторые стороны бизнес-процессов. По всей сути это не требование к ПО, но оно служит источником некоторых типов требований к ПО.
Ограничение	Ограничение на выбор вариантов, доступных разработчику при проектировании и разработке проектов.

Продолжение табл. 1

Понятие	Определение
Внешнее требование к интерфейсу	Описание взаимодействия между ПО и пользователем, другой программой системой или устройством.
Характеристика	Одна или несколько логически связанных возможностей системы, которые представляют ценность для пользователя и описаны рядом функциональных требований.
Функциональное требование	Описание требуемого поведения системы в определенных условиях.
Нефункциональное требование	Описание свойства или особенности, которым должна обладать система, или ограничение, которое должна соблюдать система.
Атрибут качества	Вид нефункционального требования, описывающего характеристики сервиса или производительности продукта.
Системное требование	Требование верхнего уровня к продукту, состоящему из многих подсистем, которые могут представлять собой ПО или совокупность ПО и оборудования
Пользовательское требование	Задача, которую определенные классы пользователей должны иметь возможность выполнять в системе, или требуемый атрибут продукта.

Рассмотрим подробнее содержание основных видов требований.

Бизнес-требования описывают основные бизнес-цели организации или клиента, на основании которых разрабатывается ПО. Также к бизнес-требованиям могут относиться устав проекта, варианты использования или документы рыночных требований.

Пользовательские требования описывают цели и задачи, которые пользователи считают необходимыми для выполнения ПО и ради которых создается данная система. Область пользовательских

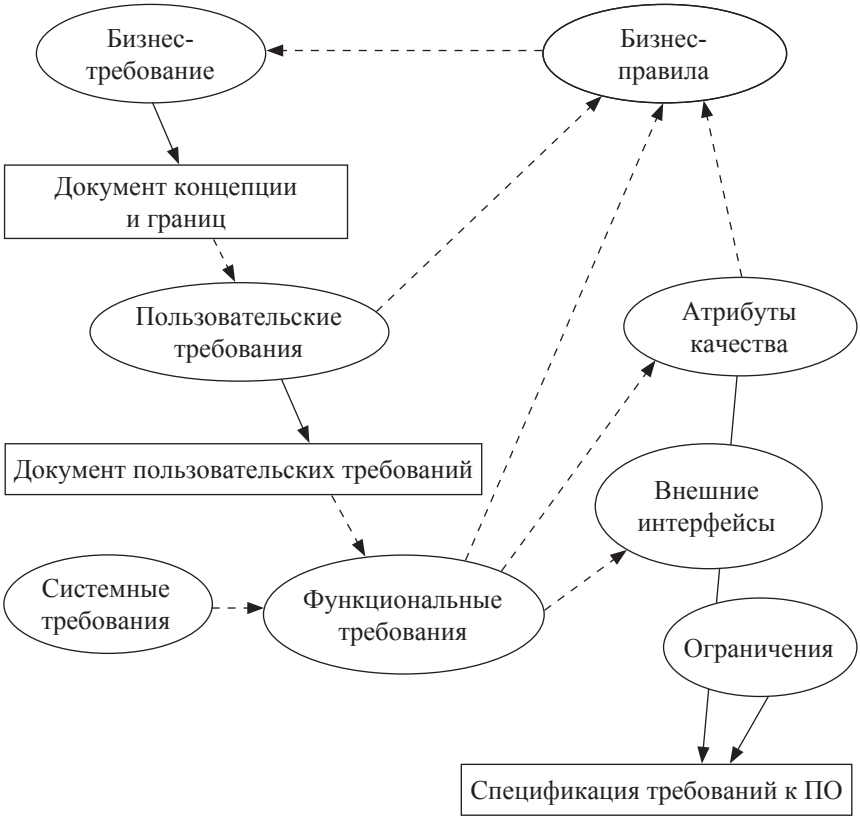


Рис. 2.1. Взаимосвязи нескольких типов информации для требований

требований также включает описания атрибутов характеристик продукта, которые важны для удовлетворения пользователей. Основным способом представления пользовательских требований являются варианты использования, пользовательские истории и таблицы «событие – отклик». В идеале эту информацию предоставляют реальные представители пользователей. Пользовательские требования описывают, какие возможности должен иметь пользователь при работе с системой.

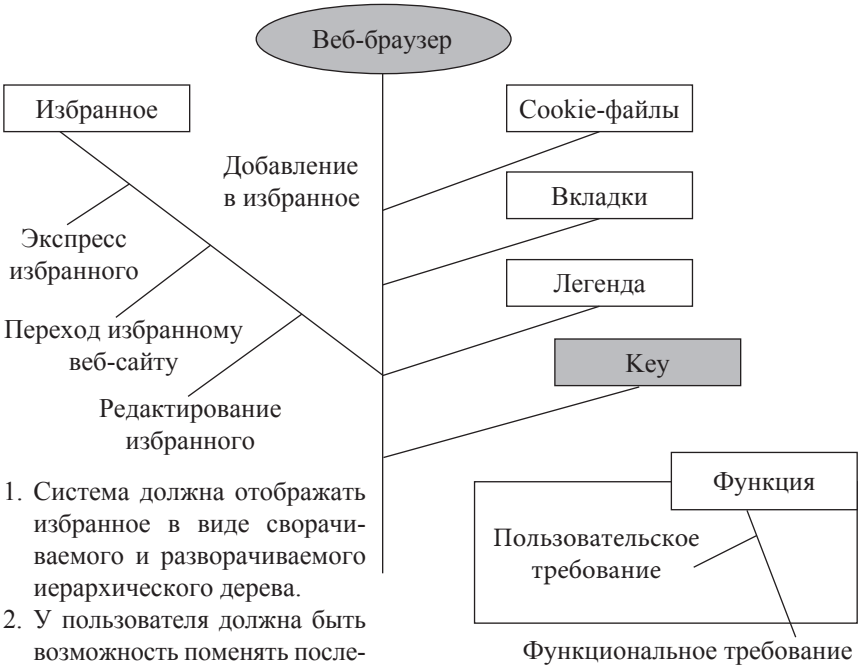
Функциональные требования определяют, каким должно быть поведение продукта в тех или иных условиях. Данные требования четко формулируют, какой продукт разработчики должны создать, чтобы пользователи смогли выполнить свои задачи (пользовательские требования) в рамках бизнес-требований. Данное соотношение между тремя уровнями требований определяет успешность реализации ПО.

Функциональные требования описываются в форме традиционных утверждений со словами «должен» или «должна»: «У пассажира должна быть возможность распечатать посадочные талоны на все рейсы, на которые он зарегистрировался».

Бизнес-аналитик документирует функциональные требования в спецификации требований к ПО и описывает их настолько подробно, насколько это необходимо в соответствии с ожидаемым поведением и свойством системы. Спецификация требований к ПО используется при разработке, тестировании, гарантии качества продукта, управлении проектом, а также на всех этапах реализации проекта. Этот артефакт называют по-разному: документ бизнес-требований, функциональная спецификация, документ требований и т.п. Спецификация требований к ПО может представлять собой отчет, сгенерированный на основе информации, хранимой в средстве управления требованиями.

Системные требования описывают требования к продукту, состоящему из многих компонентов или подсистем (ISO/IEC/IEEE 2011). Таким образом, система ПО включает в себя программное обеспечение, оборудование, персонал, обслуживающие процессы.

Бизнес-правила включают корпоративную политику, правительственные постановления, отраслевые стандарты и вычислительные алгоритмы. Бизнес-правила не являются исключительно требованиями к ПО, потому что они находятся за пределами любой системы ПО. Однако они часто налагают ограничения, определяя, какими функциями должна обладать система, подчиняющаяся соответствующим правилам. Иногда, как в случае с корпоративными политиками безопасности, бизнес-правила становится источником атрибутов качества, которые реализуются в функциональности.



1. Система должна отображать избранное в виде сворачиваемого и разворачиваемого иерархического дерева.
2. У пользователя должна быть возможность поменять последовательность избранного.
3. Система должна отображать свойства избранного.
4. У пользователя должна быть возможность поменять имя, адрес и описание избранного сайта.

Рис. 2.2. Взаимоотношения между функциями и пользовательскими требованиями

Нефункциональные требования. Атрибуты качества еще называют параметрами качества, требованиями по уровню обслуживания и т.п. Они представляют собой описание различных измерений характеристик продукта, которые важны для пользователей и разработчиков, таких как производительность, доступность и переносимость. Другие нефункциональные требования описывают внешние интерфейсы между системой и внешним миром. Речь идет о подключе-

ниях к другим программным системам, аппаратным устройствам и пользователям, а также коммуникационные интерфейсы. Ограничения проектирования и реализации накладывают границы на возможности выбора разработчика при проектировании продукта.

Характеристика — это набор логически связанных требований, которые представляют ценность для пользователя и удовлетворяют бизнес-цели. Желательные характеристики продукта, которые перечисляет клиент, не эквивалентны тем, что входят в список необходимых для решения задач пользователей. В качестве примеров характеристик продуктов можно привести избранные станицы или закладки веб-браузера, средства проверки орфографии, запись макрокоманды, автоматическое обновление определений вирусов в антивирусной программе. Характеристики могут охватывать множество пользовательских требований, и для каждого варианта необходимо, чтобы множество функциональных требований было реализовано для выполнения пользователем его задач. На рис. 2.2 представлена модель анализа, которая показывает, как функцию можно представить в виде иерархии более мелких функций, которые связаны с конкретными пользовательскими требованиями и ведут к определению наборов функциональных требований [14].

Проверка грамматики имеет множество индивидуальных требований, которые имеют дело с такими операциями, как поиск и выделение слов с ошибками, авто-исправление, отображение вариантов замены и замена слова с ошибкой корректным вариантом по всему тексту. Требования по удобству использования определяют, как нужно локализовать ПО для использования с различными языками и наборами символов.

2.2. Взаимодействие участников в процессе разработки требований

Различными организациями используются различные названия для ролей участников данного процесса. Имена ролей часто различаются в зависимости от того, является ли подразделение, разраба-

тывающее продукт, внутренним отделом организации или компанией, создающей ПО для коммерческого использования.

На основе выявленной бизнес-потребности, требования рынка или интересной компетенции нового продукта менеджеры и сотрудники отдела маркетинга определяют бизнес-требования для ПО, которые помогут компании работать эффектнее или успешно конкурировать на рынке. В корпоративной среде после этого аналитики обычно работают с представителями пользователей для определения пользовательских требований. В компаниях, разрабатывающих коммерческие продукты, часто имеется должность так называемого менеджера продукта, который определяет, какие функции должны включаться в новый продукт. Каждое пользовательское требование должно быть сопоставлено бизнес-требованию. На основе пользовательских требований аналитик или менеджер продукта определяет функции, которые дадут возможность пользователям выполнять их задачи.

Функциональные и нефункциональные требования являются основой для создания разработчиками ПО в соответствии с требуемой функциональностью, не выходя за рамки налагаемых ограничений. Тестировщики определяют, как проверять правильность реализации требований.

На рис. 2.3 представлено взаимодействие различных заинтересованных лиц в процессе выявления трех уровней требований к программному продукту.

Что касается требований к самому проекту, то здесь, как правило, создаются требования других типов: документ, где описана среда разработки, ограничения бюджета, руководство пользователя или требования для выпуска продукта и продвижения его в поддерживаемую среду. Спецификация требований содержит требования и не содержит деталей дизайна или реализации (кроме известных ограничений), данных о планировании проекта, сведений о тестировании и тому подобной информации. К требованиям проекта относятся:

- физические ресурсы, необходимые команде разработки, такие как рабочие станции, специальные аппаратные устройства, те-

Рис. 2.3. Пример участия различных заинтересованных лиц в разработке требований

стовые лаборатории, средства и оборудование тестирования, командные комнаты и оборудование для видеоконференций;
- потребность в обучении персонала;
- пользовательская документация, включая обучающие материалы, пособия, справочные руководства, а также информация о выпусках ПО;
- документация для поддержки, такая как ресурсы службы технической поддержки, а также информация о техническом обеспечении и обслуживании аппаратных устройств;
- инфраструктура изменений, которые необходимо внести в рабочую среду;
- требования и процедуры для выпуска продукта, установки в рабочей среде, конфигурирования и тестирования;
- тестирование и продукты для перехода со старой на новую систему, например требования по преобразованию данных, по настройке безопасности — эти требования иногда называют требованиями по переходу [13];
- требования по спецификации продукта и его соответствия требованиям регулирующих органов;
- скорректированная политика, процессы, организационные структуры и аналогичные документы;
- сорсинг, приобретение и лицензирование ПО сторонних производителей и компонентов оборудования;
- требования по бета-тестированию, производству, упаковке, маркетингу и дистрибуции;
- соглашения об уровне обслуживания с клиентами;
- правовые требования по интеллектуальной собственности, связанной с разрабатываемым ПО (патенты, товарные знаки или авторское право).

Определение этих требований к проекту является совместной ответственностью бизнес-аналитика и менеджера проекта. Они часто возникают при сборе требований к продукту.

Приемы формулирования требований представлены в приложении 1.

2.3. Процесс разработки требований

Поэтапный процесс разработки требований представлен на рис. 2.4.

Аналитик на этапе выявления требований обрабатывает информацию, полученную от клиентов, производит классификацию на категории и соотносит потребности клиента с возможными программными требованиями (этап анализа). В процессе анализа мо-

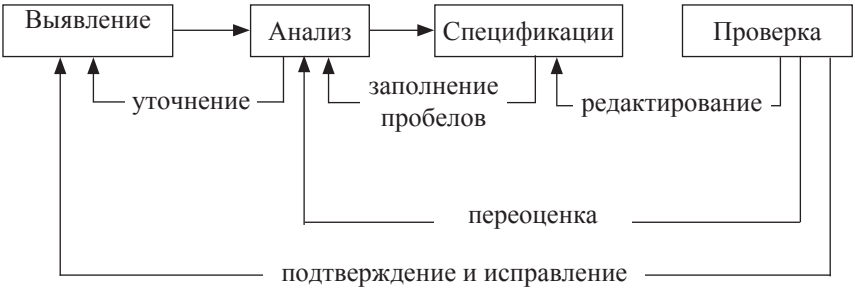


Рис. 2.4. Процесс разработки требований

жет возникнуть необходимость в уточнении некоторых требований, тогда аналитик будет возвращаться назад на этап выявления требований. Далее аналитик структурирует информацию от пользователей и выведенные требования — это этап спецификации требований. На данном этапе также может возникнуть необходимость в дополнительном анализе. Затем происходит процесс подтверждения представленных требований заинтересованными лицами (этап проверки), в результате которого также может возникнуть необходимость в исправлениях и доработках. После этого аналитик переходит к следующему этапу проекта и весь цикл повторяется. Такой итеративный процесс продолжается на всем протяжении разработки требований и возможно — как в проектах гибкой разработки — на протяжении всего времени проекта.

Стоит отметить разнообразие проектов по разработке ПО, отсутствие единого, шаблонного подхода к созданию требований. На

рис. 2.5 показан общий процесс создания требований, который с разумными исправлениями подойдет для большинства проектов.

Как правило, действия выполняются по порядку, однако сам процесс не является строго последовательным. Первые семь действий обычно однократно выполняются на ранних стадиях работы над проектом. Остальные необходимы для каждого очередного выпуска или этапа работы над проектом. Многие из этих действий могут выполняться итеративно и попеременно. Например, шаги 8, 9 и 10 можно осуществлять небольшими порциями, выполняя пересмотр (шаг 12) после каждой итерации.

Пятым подразделом разработки требований, не вошедшим в общую схему (рис. 2.4), является управление требованиями. К управлению требованиями относятся приемы работы с уже готовыми требованиями. Эти приемы включают управление версиями и определение базовых требований, управление изменениями, отслеживание состояния требований и отслеживание связей требований с другими элементами системы.

Управление требованиями — это деятельность низкой интенсивности, происходящая на всем протяжении проекта.

Пример одного из процессов разработки требований приведен на рис. 2.5.

Рассмотрим подробнее основные этапы итеративного процесса разработки требований.

Выявление требований

На данном этапе процесса разработки требований можно выделить следующие уровни.

1. Определение концепции продукта и границ проекта. Границы проекта определяют, что следует реализовать в данной версии, а что — в последующих. Концепция и границы являются хорошей базой для оценки предлагаемых требований.
2. Определение классов пользователей и их характеристик. Для определения требований необходимо создать классы поль-

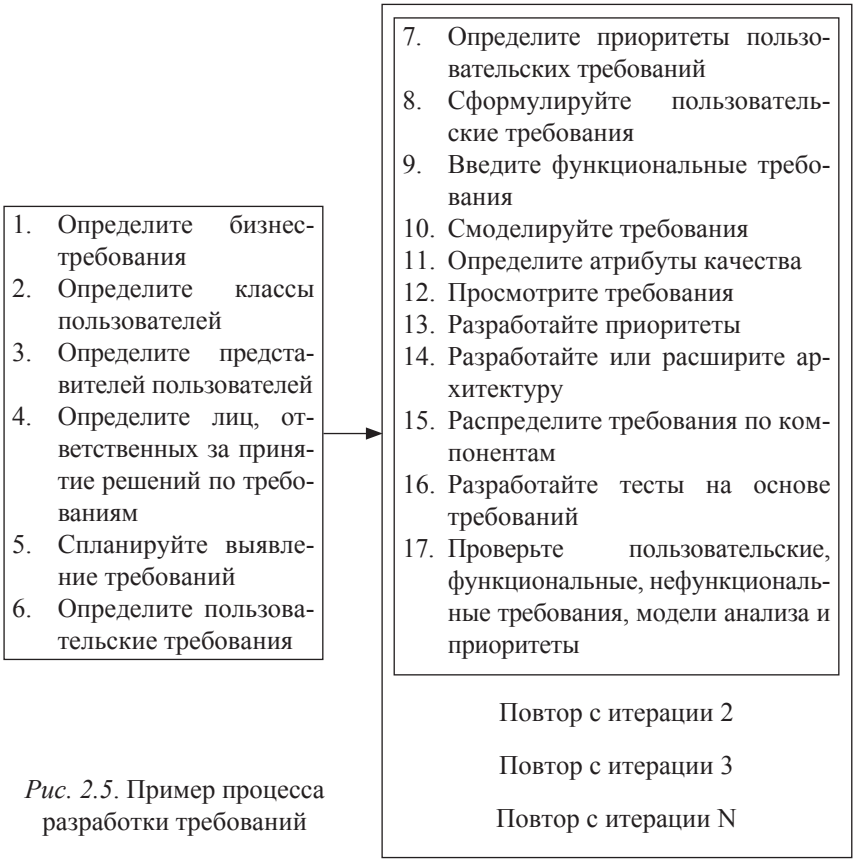


Рис. 2.5. Пример процесса разработки требований

зователей или сгруппировать их в группы по частоте работы с ПО, используемым функциям, уровню привилегий и опыту работы и т.д.

3. Выбор сторонника продукта в каждом классе пользователей. Это человек, который сможет точно передавать настроения и нужды клиентов. Он представляет потребности определенного класса пользователей и принимает решения от их лица.
4. Проведение фокус-групп типичных пользователей. Фокус-группы особенно важны при разработке коммерческих про-

дуктов, когда приходится иметь дело с большой и разнородной клиентской базой. В отличие от сторонников продукта у фокус-групп обычно нет полномочий на принятие решений.

5. Работа с пользователями для выяснения назначения продукта. Пользовательские требования могут выражаться в форме сценариев использования, задач или пользовательских историй.
6. Проведение интервью для выявления требований. Интервью можно проводить в формате «один на один» или с небольшой группой заинтересованных лиц. Это эффективный способ выявления требований.
7. Проведение совместных семинаров. Совместные семинары по выявлению требований для аналитиков и пользователей представляют отличный способ выявить нужды пользователей.
8. Наблюдение за пользователями на рабочих местах. Простые диаграммы рабочих потоков, а также диаграммы потоков данных позволяют выяснить этапы и принимаемые решения. Документируя ход бизнес-процесса, можно точнее определить требования к системе.
9. Анализ документации.
10. Повторное использование требований. Если необходимая клиенту функциональность аналогична уже реализованной в другом продукте, можно в проекте использовать существующие компоненты.

Анализ требований

Целью этапа анализа требований является достаточно качественное и подробное описание требования, которое позволит менеджерам реалистично оценить все затраты на проект, а техническому персоналу приступить к разработке и тестированию ПО. Данный этап может включать следующие уровни.

1. Моделирование среды приложения (создание контекстной карты, карты экосистемы). Контекстная диаграмма является простой и эффективной моделью анализа, отображающей ме-

сто новой системы в соответствующей среде. Она определяет границы и интерфейсы между разрабатываемой системой и сущностями, внешними для этой системы, например пользователями, устройствами и прочими информационными системами. Карта экосистемы показывает взаимодействие систем друг с другом, а также природу их взаимосвязей.

2. Создание прототипа пользовательского интерфейса и технических прототипов. Оценка прототипа поможет разработчикам и пользователям достичь взаимопонимания по решаемой проблеме, а также проверить требования.
3. Анализ осуществимости требований. Аналитик совместно с разработчиком обязан определить возможности удовлетворения каждого требования к системе при разумных затратах и с приемлемой производительностью в предполагаемой среде. Это позволит заинтересованным лицам понять риски, связанные с реализацией каждого требования.
4. Определение приоритетов требований. Важным этапом является определение приоритетов требований, чтобы обеспечить необходимую функциональность разрабатываемой системы. Необходимо использовать аналитический подход и расставить относительные приоритеты реализации функций продукта, решаемых задач, пользовательских историй или отдельных требований. На основании приоритетов устанавливается, в какой версии будет реализована та или иная функция или набор требований.
5. Создание словаря данных. Словарь данных содержит определения всех элементов и структур данных, что позволит всем участникам проекта использовать согласованные определения данных. На стадии работы над требованиями словарь должен содержать определения элементов данных, относящихся к предметной области, чтобы клиентам и разработчикам было проще общаться.
6. Моделирование требований Модель анализа представляет собой диаграмму, которая визуально отображает требования.

К данным моделям относятся диаграммы потоков данных, диаграммы «сущность-связь», диаграммы перехода состояний, таблицы состояний, карты диалоговых окон, деревья решений и др. [13].

7. Анализ интерфейсов между системой и внешним миром.
8. Распределение требований по подсистемам. Требования к сложному продукту, включающему несколько подсистем, следует соразмерно распределять между программными, аппаратными и операторскими подсистемами и компонентами.

Спецификация требований

Целью этапа спецификации требований является документирование различных требований при помощи специальных средств. Пользовательские требования обычно представляют в виде вариантов использования. Подробные функциональные и нефункциональные требования к ПО фиксируются в спецификации требований к ПО.

Рассмотрим содержание основных уровней данного этапа.

1. Внедрение шаблонов документов требований. Создание стандартных шаблонов для документирования требований к ПО, таких как документ концепции и границ, шаблон варианта использования, позволит фиксировать описания разной информации, касающейся требований.
2. Определение источников требований. Для описания требований необходимо определить источники, которыми могут быть варианты использования или другая информация от пользователей, системное требование высокого уровня или бизнес-правило.
3. Присвоение уникальных идентификаторов всем требованиям.
4. Документирование бизнес-правил.
5. Определение нефункциональных требований.

Проверка требований

Данный этап предполагает проверку корректности всех положений требований и включает в себя следующие уровни.

1. Рецензирование требований.
2. Тестирование требований.
3. Определение критериев приемки. Критерии представляют собой приемочные тесты, основанные на требованиях и отслеживающие возможность удовлетворения определенным нефункциональным требованиям, отслеживания открытых дефектов и проблем.
4. Моделирование требований. Моделирование представляет собой уровень, который позволяет аналитику в процессе взаимодействия с пользователями быстро построить макет системы.

Управление требованиями

Начальные требования неизбежно корректируются в процессе работы клиентами, менеджерами, специалистами по маркетингу, разработчиками. Для эффективного управления требованиями необходимы выверенные способы управления конфигурацией. Для управления требованиями можно использовать те же утилиты, что и для управления версиями, которые вы используете для управления базой кода. Рассмотренный итеративный процесс разработки требований в каждом конкретном случае будет представлять индивидуальное решение.

Процесс управления требованиями включает действия, необходимые для обеспечения целостности, точности и своевременности обновления соглашения о требованиях в ходе проекта. На рис. 2.6 представлены основные операции, входящие в состав четырех базовых функций: управление версиями, управление изменениями, отслеживание состояния требований и отслеживание связей требований.

Рис. 2.6. Базовые операции управления требованиями

При подготовке к процессу управления требованиями необходимо определить следующие параметры:

- инструменты, приемы и соглашения, определяющие правила создания различных версий требований;
- наборы базовых версий требований;
- методы внесения изменений в существующие требования;
- степень влияния предполагаемых изменений в требования;
- атрибуты требований и процедуры отслеживания состояний требований;

- определить ответственных лиц за обновление информации об изменении требований.

2.4. Разработка технического задания на проектирование информационных систем

Техническое задание на проектирование является основным документом, регламентирующим процесс и результаты проектирования. Начальным этапом для разработки технического задания является предпроектное исследование.

Целью предпроектных исследований является преобразование общих нечетких знаний о предназначении будущего программного обеспечения в сравнительно точные требования к нему. На данном этапе необходимо провести анализ предметной области и рассмотреть два варианта неопределенности:

- неизвестны методы решения формулируемой задачи — такого типа неопределенности обычно возникают при решении научно-технических задач;
- неизвестна структура автоматизируемых информационных процессов — обычно встречается при построении автоматизированных систем управления предприятием

Для первого случая во время предпроектных исследований определяется возможность решения поставленной задачи, а также рассматриваются методы, позволяющие получить требуемый результат, что может потребовать соответствующих научных исследований как фундаментального, так и прикладного характера, разработки и исследования новых моделей объектов реального мира.

Для второго случая рассматривается и определяется:

- структура и взаимосвязь автоматизируемых информационных процессов;
- основные функции разрабатываемого ПО; внешние условия его функционирования и особенности его интерфейсов как с пользователями, так и — при необходимости — с аппаратной частью;

- основные требования к программному и информационному обеспечению, необходимые аппаратные ресурсы, требования к базам данных и физические характеристики программных компонент.

Результатом предпроектных исследований предметной области является разработка технического задания.

Техническое задание представляет собой документ, в котором сформулированы основные цели разработки, требования к программному продукту, определены сроки и этапы разработки и регламентирован процесс приемо-сдаточных испытаний. В разработке технического задания участвуют как представители заказчика, так и представители исполнителя.

В основе этого документа лежат исходные требования заказчика, анализ передовых достижений техники, результаты выполнения научно-исследовательских работ, предпроектных исследований, научного прогнозирования и т.п.

Рассмотрим основные факторы, которые определяют характеристики разрабатываемого программного обеспечения:

- исходные данные и требуемые результаты, которые определяют функции программы или системы;
- среда функционирования (программная и аппаратная) — может быть задана, а может выбираться для обеспечения параметров, указанных в техническом задании;
- возможное взаимодействие с другим программным обеспечением и/или специальными техническими средствами — также может быть определено, а может выбираться, исходя из набора выполняемых функций.

Разработка технического задания выполняется в следующей последовательности. Прежде всего устанавливают набор выполняемых функций, а также перечень и характеристики исходных данных. Затем определяют перечень результатов, их характеристики и способы представления.

Далее уточняют среду функционирования программного обеспечения: конкретную комплектацию и параметры технических средств,

версию используемой операционной системы и, возможно, версии и параметры другого установленного программного обеспечения, с которым предстоит взаимодействовать будущему программному продукту. В случаях, когда разрабатываемое программное обеспечение собирает и хранит некоторую информацию или включается в управление каким-либо техническим процессом, необходимо также четко регламентировать действия программы при сбоях оборудования и энергоснабжения.

На техническое задание существует стандарт ГОСТ 19.201–78 «Техническое задание. Требования к содержанию и оформлению».

В соответствии с этим стандартом техническое задание должно содержать следующие разделы:

1. ВВЕДЕНИЕ
2. ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ
3. НАЗНАЧЕНИЕ
4. ТРЕБОВАНИЯ К ПРОГРАММЕ ИЛИ К ПРОГРАММНОМУ СРЕДСТВУ
   4.1. Требования к функциональным характеристикам
   4.4.1. Выполняемые функции
   4.1.2. Исходные данные
   4.2. Требования к надёжности
   4.3. Требования к составу и параметрам технических средств
   4.4. Требования к информационной и программной совместимости
5. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ.

При необходимости допускается в техническое задание включать приложения.

Рассмотрим более подробно содержание каждого раздела.

Введение должно включать наименование и краткую характеристику области применения программы или программного продукта, а также объекта (например, системы) в котором предполагается их использовать.

Основное назначение введения — продемонстрировать актуальность данной разработки и показать, какое место эта разработка занимает в ряду подобных.

Раздел **Основания для разработки** должен содержать наименование документа, на основании которого ведется разработка, организации, утвердившей данный документ, и наименование или условное обозначение темы разработки. Таким документом может служить план, приказ, договор и т.п.

Раздел **Назначение разработки** должен содержать описание функционального и эксплуатационного назначения программного продукта с указанием категорий пользователей.

Раздел **Требования к программе или программному изделию** должен включать следующие подразделы:

- требования к функциональным характеристикам;
- требования к надежности;
- условия эксплуатации;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- требования к маркировке и упаковке;
- требования к транспортированию и хранению;
- специальные требования.

Наиболее важным из перечисленных выше является подраздел «Требования к функциональным характеристикам».

В этом разделе должны быть перечислены выполняемые функции и описаны состав, характеристики и формы представления исходных данных и результатов. В этом же разделе при необходимости указывают критерии эффективности: максимально допустимое время ответа системы, максимальный объем используемой оперативной и/или внешней памяти и др.

Примечание. Если разработанное программное обеспечение не будет выполнять указанных в техническом задании функций, то оно считается не соответствующим техническому заданию, т.е. неправильным с точки зрения критериев качества. Универсальность будущего продукта также обычно специально не оговаривается, но подразумевается.

В подразделе «Требования к надежности» указывают уровень надежности, который должен быть обеспечен разрабатываемой сис-

темой, и время восстановления системы после сбоя. Для систем с обычными требованиями к надежности в этом разделе иногда регламентируют действия разрабатываемого продукта по увеличению надежности результатов (контроль входной и выходной информации, создание резервных копий промежуточных результатов и т.п.).

В подразделе «Условия эксплуатации» указывают особые требования к условиям эксплуатации: температуре окружающей среды, относительной влажности воздуха и т.п. Как правило, подобные требования формулируют, если разрабатываемая система будет эксплуатироваться в нестандартных условиях или использует специальные внешние устройства, например для хранения информации. Здесь же указывают вид обслуживания, необходимое количество персонала и его квалификация. В противном случае допускается указывать, что требования не предъявляются.

В подразделе «Требования к составу и параметрам технических средств» указывают необходимый состав технических средств с указанием их основных технических характеристик: тип микропроцессора, объем памяти, наличие внешних устройств и т.п. При этом часто указывают два варианта конфигурации: минимальный и рекомендуемый.

В подразделе «Требования к информационной и программной совместимости» при необходимости можно задать методы решения, определить язык или среду программирования для разработки, а также используемую операционную систему и другие системные и пользовательские программные средства, с которыми должно взаимодействовать разрабатываемое программное обеспечение. В этом же разделе при необходимости указывают, какую степень защиты информации необходимо предусмотреть.

В разделе **Требования к программной документации** указывают необходимость наличия руководства программиста, руководства пользователя, руководства системного программиста, пояснительной записки и т.п. На все эти типы документов также существуют ГОСТы.

В разделе **Технико-экономические показатели** рекомендуется указывать ориентировочную экономическую эффективность, пред-

полагаемую годовую потребность и экономические преимущества по сравнению с существующими аналогами.

В разделе **Стадии и этапы разработки** указывают стадии разработки, этапы и содержание работ с указанием сроков разработки и исполнителей.

В разделе **Порядок контроля и приемки** указывают виды испытаний и общие требования к приемке работы.

В приложениях при необходимости приводят: перечень научно-исследовательских работ, обосновывающих разработку; схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые следует использовать при разработке.

В зависимости от особенностей разрабатываемого продукта разрешается уточнять содержание разделов, т.е. использовать подразделы, вводить новые разделы или объединять их. В случаях, если какие-либо требования, предусмотренные техническим заданием, заказчик не предъявляет, следует в соответствующем месте указать «Требования не предъявляются».

3. ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ UML

3.1. Назначение и описание языка UML

Язык UML (Unified Modeling Language) — это графический язык моделирования общего назначения, предназначенный для спецификации, визуализации, проектирования и документирования всех артефактов, создаваемых при разработке программных систем [16–18].

В настоящее время разработчиками широко используются методы объектно-ориентированного анализа, дизайна и разработки. В данной методологии объекты (objects) предметной области разрабатываемого ПО соотносятся с объектами реального мира. Они представляют отдельные экземпляры, выведенные из общего шаблона — класса (class). В описания классов входят атрибуты (данные) и действия, которые можно выполнять с этими атрибутами. Диаграмма классов (classdiagram) — это графический способ отобразить классы, идентифицированные в ходе объектно-ориентированного анализа, и взаимоотношения между ними.

Для продуктов, разработанных с помощью объектно-ориентированных методов, не требуются какие-то особые приемы разработки требований. Однако при использовании объектно-ориентированных методов для разработки системы анализ требований стоит начать с определения классов доменов, их атрибутов и поведения. Данный подход облегчит переход от анализа к дизайну.

Унифицированный язык моделирования (Unified Modeling Language, UML) является стандартным языком объектно-ориентированного моделирования [15, 18, 19, 20]. UML в первую очередь используется для создания моделей дизайна. Возможно несколько вариантов использования UML при проведении анализа требований на уровне абстракции:

- диаграммы классов, которые отображают классы объектов предметной области: их атрибуты, свойства, а также отношения между классами;
- диаграммы вариантов использования — предназначены для отображения отношений между действующими лицами, являющимся внешними по отношению к системе, и вариантов использования, с которыми они взаимодействуют;
- диаграммы действий предназначены для отображения взаимодействия различных потоков или моделирования бизнес-процессов;
- диаграммы состояний представляют различные состояния, которые могут принимать объекты системы или данные, а также разрешенные переходы между состояниями.

Язык UML по своей сути является формальным искусственным языком, Для описания формальных искусственных языков, необходимо наличие как минимум трех составляющих:

– синтаксиса, т.е. правил составления конструкций языка,
– семантики, т.е. правил приписывания смысла конструкциям языка,
– прагматики, т.е. правил использования конструкций языка для достижения конкретных целей.

Как формальный искусственный язык UML имеет все эти составляющие, хотя эти части имеют иные названия и описаны другими способами, чем в текстовых языках программирования. Это объясняется тем, что UML — язык графический, а не текстовый, и тем, что UML — язык не программирования, а именно моделирования.

Термин «моделирование» имеет несколько значений и способов употребления. В отношении разработки программного обеспечения так сложилось, что результаты фаз анализа и проектирования,

оформленные средствами определенного языка, принято называть моделью. Деятельность по составлению моделей естественно назвать моделированием. Именно в этом смысле UML является языком моделирования.

Модель UML — это описание объекта или явления.

Рассмотрим основные этапы разработки моделей с использованием языка UML.

Спецификация

Одним из ключевых этапов разработки приложения является определение того, каким требованиям должно удовлетворять разрабатываемое приложение. В результате этого этапа появляется формальный или неформальный документ, который называют поразному, имея в виду примерно одно и то же: постановка задачи, требования, техническое задание, внешние спецификации и др.

Необходимо принимать во внимание три толкования спецификаций:

– понимание источника спецификации (например, заказчика),
– понимание потребителя спецификации (например, разработчика),
– понимание, объективно обусловленное природой специфицируемого объекта.

Эти три трактовки спецификаций могут не совпадать, и, как показывает практика, часто не совпадают, причем значительно.

Основное назначение UML — предоставить, с одной стороны, достаточно формальное, с другой стороны, достаточно удобное, и, с третьей стороны, достаточно универсальное средство, позволяющее до некоторой степени снизить риск расхождений в толковании спецификаций.

Визуализация

Модели UML допускают визуальное представление в форме картинок, причем эти картинки наглядны, интуитивно понятны, практически однозначно интерпретируются и легко составляются.

Таким образом, второе по важности назначение UML состоит в том, чтобы служить адекватным средством коммуникации между людьми. Разумеется, наглядность визуализации моделей UML имеет значение, только если они должны составляться или восприниматься человеком — это назначение UML не имеет отношения к компьютерам.

Проектирование

В оригинале данное назначение UML определено с помощью слова construct, которое мы передаем осторожным термином «проектирование». Речь идет о том, что UML предназначен не только для описания абстрактных моделей приложений, но и для непосредственного манипулирования артефактами, входящими в состав этих приложений, в том числе такими, как программный код. Таким образом, одним из назначений UML является создание таких моделей, для которых возможна автоматическая генерация программного кода (или фрагментов кода) соответствующих приложений. Более того, природа моделей UML такова, что возможен и обратный процесс: автоматическое построение модели по коду готового приложения. Инструменты, поддерживающие UML, постоянно совершенствуются, в перспективе третье предназначение UML может выйти на первое место.

Документирование

Модели UML являются артефактами, которые можно хранить и использовать как в форме электронных документов, так и в виде твердой копии. В последних версиях UML для достижения более полного соответствия этому назначению сделано довольно много. В частности, специфицировано представление моделей UML в форме документов в формате XML, что обеспечивает практическую интероперабельность при работе с моделями.

3.2. Способы использования UML

Рассмотрим основные способы использования UML для решения различных задач.

Рисование картинок. Графические средства UML можно и нужно использовать безотносительно ко всему остальному. Даже рисование диаграмм карандашом на бумаге позволяет упорядочить мысли и зафиксировать для себя существенную информацию о моделируемом приложении или иной системе.

Обмен информацией. Сообщество людей, применяющих и понимающих UML, стремительно растет. Если вы будете использовать UML, то вас будут понимать другие, и вы будете понимать их.

Спецификация систем. Это важнейший способ использования UML. И хотя не во всех случаях UML оказывается абсолютно адекватным средством спецификации, мы надеемся, что по мере развития языка все меньше будет оставаться таких исключений, где UML неприменим.

Повторное использование архитектурных решений. Повторное использование ранее разработанных решений — ключ к повышению эффективности. К сожалению, модели UML пока что повторно используются в весьма ограниченных масштабах.

Генерация кода. Генерировать код нужно и можно, но возможности имеющихся инструментов не стоит переоценивать.

Имитационное моделирование. Возможности построения моделей UML, из которых путем вычислительных экспериментов можно было бы извлекать информацию о моделируемом объекте, пока что уступают возможностям специализированных систем, сконструированных для этих целей.

3.3. Модель UML и ее элементы

Модель UML (UML model) — это совокупность конечного множества конструкций языка, главные из которых — это сущности и отношения между ними. Сами сущности и отношения модели являются экземплярами метаклассов метамодели.

Сущности

Сущности являются основными объектно-ориентированными элементами языка UML, позволяющими создавать корректные модели. Сущности являются ключевыми абстракциями языка, отношения служат для связывания сущностей друг с другом, а диаграммы группируют коллекции сущностей.

В UML существует четыре группы сущностей: структурные, поведенческие, группирующие и аннотационные.

Структурные сущности используются для описания структуры. Обычно к структурным сущностям относят следующие.

Объект (object) — сущность, обладающая уникальностью и инкапсулирующая в себе состояние и поведение.

Класс (class) — описание множества объектов с общими атрибутами, определяющими состояние, и операциями, определяющими поведение.

Интерфейс (interface) — именованное множество операций, определяющее набор услуг, которые могут быть запрошены потребителем и предоставлены поставщиком услуг.

Кооперация (collaboration) — совокупность объектов, которые взаимодействуют для достижения некоторой цели.

Действующее лицо (actor) — сущность, находящаяся вне моделируемой системы и непосредственно взаимодействующая с ней.

Компонент (component) — модульная часть системы с четко определенным набором требуемых и предоставляемых интерфейсов.

Артефакт (artifact) — элемент информации, который используется или порождается в процессе разработки программного обеспечения. Другими словами, артефакт — это физическая единица реализации, получаемая из элемента модели (например, класса или компонента).

Узел (node) — вычислительный ресурс, на котором размещаются и при необходимости выполняются артефакты.

Рассмотрим ***поведенческие сущности*** языка UML.

Состояние (state) — период в жизненном цикле объекта, находясь в котором объект удовлетворяет некоторому условию и осуществляет собственную деятельность или ожидает наступления некоторого события.

Деятельность (activity) можно считать частным случаем состояния, который характеризуется продолжительными (по времени) неатомарными вычислениями.

Действие (action) — примитивное атомарное вычисление.

Это только основная часть поведенческих сущностей: состояния бывают самые разнообразные. Кроме того, при моделировании поведения используются вспомогательные сущности, которые сосуществуют только вместе с перечисленными основными.

Особняком стоит сущность – вариант использования, ей присущи как структурные, так и поведенческие аспекты.

Вариант использования (use case) — множество сценариев, объединенных по некоторому критерию и описывающих последовательности производимых системой действий, доставляющих значимый для некоторого действующего лица результат.

Группирующая сущность в UML одна, зато универсальная — пакет.

Пакет (package) — группа элементов модели (в том числе пакетов).

Аннотационная сущность тоже одна — примечание (comment), зато в нее можно поместить все что угодно, так как содержание примечания UML не ограничивает.

Приведенная классификация не является исчерпывающей. У перечисленных сущностей есть различные частные случаи и варианты, которые будут рассмотрены ниже.

Отношения

В UML используются четыре основных типа отношений:

- зависимость;
- ассоциация;

- обобщение (generalization);
- реализация.

Зависимость (dependency) — это наиболее общий тип отношений между двумя сущностями, этот тип указывает на то, что изменение независимой сущности каким-то образом влияет на зависимую сущность.

Ассоциация (association) используется для описания непосредственно связи одной сущности с другой (данный тип отношений не обязательно является бинарным).

Обобщение (realization) — это отношение между двумя сущностями, одна их которых является частным случаем другой.

Реализация (realization) — данное отношение указывает на то, что одна сущность является реализацией другой.

Диаграммы

Диаграмма (diagram) — это графическое представление некоторой части графа модели. Основной набор диаграмм, используемых для разработки, называется каноническими диаграммами. Иерархия типов диаграмм для UML 1 представлена на рис. 3.1. На диаграммах могут присутствовать дополнительные конструкции языка.

Основные типы диаграмм:

- диаграмма прецедентов (Use Case diagram);
- диаграмма классов (Class diagram);
- диаграмма объектов (Object diagram);
- диаграмма состояний (State chart diagram);
- диаграмма деятельности (Activity diagram);
- диаграмма последовательности (Sequence diagram);
- диаграмма кооперации (Collaboration diagram);
- диаграмма компонентов (Component diagram);
- диаграмма размещения (Deployment diagram).

Рассмотрим подробнее представленные диаграммы.

Диаграммы прецедентов (вариантов использования) предназначены для отображения отношений между действующими лицами,

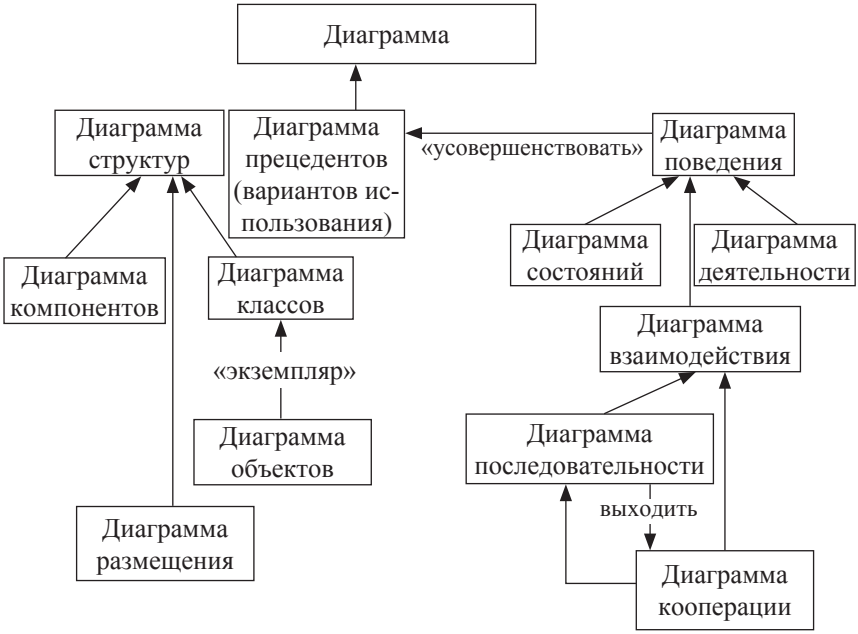


Рис. 3.1. Иерархия типов диаграмм для UML

являющимися внешними по отношению к системе, и вариантов использования, с которыми они взаимодействуют. Это наиболее общее представление функциональных требований к системе.

Диаграмма вариантов использования включает два типа сущностей: варианты использования и действующие лица. Существует 4 основных типа отношений между данными сущностями:

- ассоциации между действующим лицом и вариантом использования;
- обобщения между действующими лицами;
- обобщения между вариантами использования;
- зависимости между вариантами использования.

Диаграммы классов отображают классы объектов предметной области: их атрибуты, свойства, а также отношения между классами. На диаграмме классов применяется один основной тип сущно-

стей — классы (включая многочисленные частные случаи классов: интерфейсы, примитивные типы, классы-ассоциации и многие другие), между которыми устанавливаются следующие основные типы отношений:

- ассоциация между классами;
- обобщение между классами;
- зависимости между классами и интерфейсами.

Диаграммы состояний представляют различные состояния, которые могут принимать объекты системы или данные, а также разрешенные переходы между состояниями. Графически диаграмма состояний представляет собой граф состояний и переходов конечного автомата. На такой диаграмме применяют один основной тип сущностей — состояния, и один тип отношений — переходы.

Диаграммы деятельности предназначены для отображения деятельности различных потоков или моделирования бизнес-процессов.

На диаграмме деятельности применяют один основной тип сущностей — действие (1), и один тип отношений — переходы (2) (передачи управления). Также используются такие конструкции, как развилки, слияния, соединения, ветвления (3), которые похожи на сущности, но таковыми на самом деле не являются.

Диаграммы последовательности предназначены для отображения взаимодействия различных потоков или моделирования бизнес-процессов.

В объектно-ориентированном программировании самым существенным во время выполнения является пересылка сообщений между взаимодействующими объектами. Именно последовательность посылок сообщений отображается на данной диаграмме, отсюда и название.

На диаграмме последовательности применяют один основной тип сущностей — экземпляры взаимодействующих классификаторов, по которым происходит обмен сообщениями. Основным отличием данного типа диаграмм является наличие отображения течения времени.

Диаграммы компонентов (component diagram) показывают взаимосвязи между модулями (логическими или физическими), из которых состоит моделируемая система.

Основной тип сущностей на диаграмме компонентов — это компоненты и интерфейсы, посредством которых указывается взаимосвязь между компонентами. На диаграмме компонентов применяются следующие отношения:

- реализации между компонентами и интерфейсами;
- зависимости между компонентами и интерфейсами.

Диаграммы размещения (deployment diagram) наряду с отображением состава и связей элементов системы показывают, как они физически размещены на вычислительных ресурсах во время выполнения.

Диаграмма размещения добавляет два типа сущностей: артефакт, который является реализацией компонента, и узел, а также отношение ассоциации между узлами, показывающее, что узлы физически связаны во время выполнения.

Диаграммы кооперации (collaboration diagram) являются подвидом диаграмм композитной структуры (composite structure diagram), которые показывают роль и взаимодействие классов в рамках кооперации. Диаграммы кооперации широко применяются при моделировании шаблонов и паттернов проектирования. Такие диаграммы могут использоваться совместно с диаграммами классов.

Диаграммы объектов (object diagram) в языке моделирования UML предназначены для демонстрации совокупности моделируемых объектов и связей между ними в фиксированный момент времени.

Диаграмма объектов описывает конкретные экземпляры объектов и напрямую соотносится с диаграммой классов, которая даёт общее представление о конфигурации системы. Она используется для документирования структур данных и создания статических снимков состояний объектов, принимая во внимание реальные экземпляры или прототипы. Динамику поведения объектов обычно изображают в виде последовательности таких диаграмм.

4. ОСНОВНЫЕ СВЕДЕНИЯ О РАБОТЕ В СРЕДЕ RATIONAL ROSE

Для автоматизации процессов проектирования и разработки программного обеспечения на компьютере используются CASE-средства (computer-aided software engineering), то есть средства автоматизации разработки программ.

Rational Rose представляет собой CASE-средство проектирования и разработки информационных систем и программного обеспечения для управления предприятиями. Как и другие CASE-средства (ARIS, BPwin, ERwin), его можно применять для анализа и моделирования бизнес процессов. Современные версии продукта Rational Rose выпускаются компанией IBM.

Принципиальное отличие Rational Rose от других средств заключается в объектно-ориентированном подходе. Графические модели, создаваемые с помощью этого средства, основаны на объектно-ориентированных принципах и языке UML. Инструменты моделирования Rational Rose позволяют разработчикам создавать целостную архитектуру процессов предприятия, сохраняя все взаимосвязи и управляющие воздействия между различными уровнями иерархии.

Элементы экрана

Пять основных элементов интерфейса Rose — это браузер, окно документации, панели инструментов, окно диаграммы и журнал (log). Их назначение заключается в следующем.

- Браузер (browser) используется для быстрой навигации по модели.
- Окно документации (documentation window) применяется для работы с текстовым описанием элементов модели.
- Панели инструментов (toolbars) применяются для быстрого доступа к наиболее распространенным командам.
- Окно диаграммы (diagram window) используется для просмотра и редактирования одной или нескольких диаграмм UML.
- Журнал (log) применяется для просмотра ошибок и отчетов о результатах выполнения различных команд.

На рис. 4.1 показаны различные части интерфейса Rose.

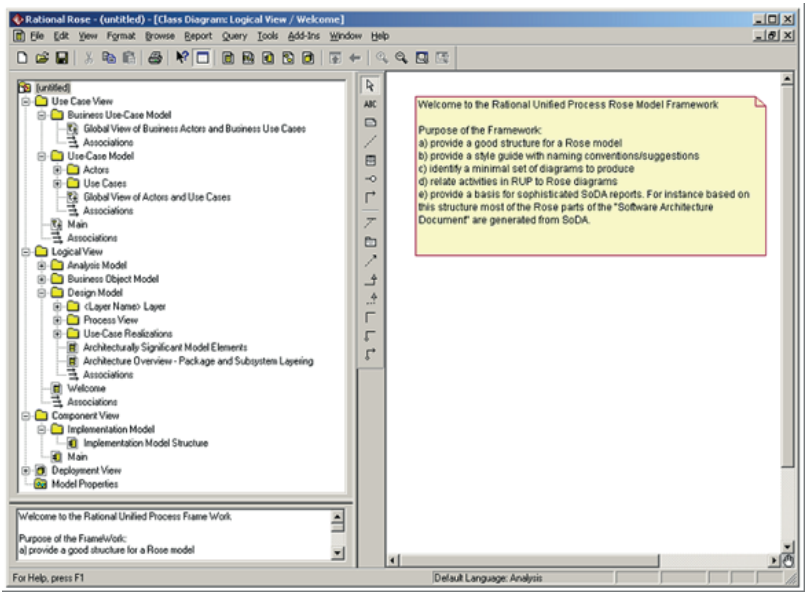


Рис. 4.1. Интерфейс среды Rose

Браузер

Браузер — это иерархическая структура, позволяющая осуществлять навигацию по модели. Все, что добавляется к ней, — действующие лица, варианты использования, классы, компоненты — будет показано в окне браузера.

С помощью браузера можно:

– добавлять к модели элементы (действующие лица, варианты использования, классы, компоненты, диаграммы и т.д.);
– просматривать существующие элементы модели;
– просматривать существующие связи между элементами модели;
– перемещать элементы модели;
– переименовывать эти элементы;
– добавлять элементы модели к диаграмме;
– связывать элемент с файлом или адресом Интернет;
– группировать элементы в пакеты;
– работать с детализированной спецификацией элемента;
– открывать диаграмму.

Браузер поддерживает четыре представления (view): представление вариантов использования, компонентов, размещения и логическое представление. Все они и содержащиеся в них элементы модели описаны ниже.

Браузер организован в древовидном стиле. Каждый элемент модели может содержать другие элементы, находящиеся ниже его в иерархии. Знак «–» около элемента означает, что его ветвь полностью раскрыта. Знак «+» означает, что его ветвь свернута.

Окно документации

С его помощью можно документировать элементы модели Rose. Например, можно сделать короткое описание каждого действующего лица. При документировании класса все, что будет написано в окне документации, появится затем как комментарий в сгенериро-

ванном коде, что избавляет от необходимости впоследствии вносить эти комментарии вручную. Документация будет выводиться также в отчетах, создаваемых в среде Rose.

Панели инструментов

Панели инструментов Rose обеспечивают быстрый доступ к наиболее распространенным командам. В этой среде существует два типа панелей инструментов: стандартная панель и панель диаграммы. Стандартная панель видна всегда, ее кнопки соответствуют командам, которые могут использоваться для работы с любой диаграммой. Панель диаграммы своя для каждого типа диаграмм UML.

Все панели инструментов могут быть изменены и настроены пользователем. Для этого выберите пункт меню Tools > Options, затем выберите вкладку Toolbars.

Чтобы показать или скрыть стандартную панель инструментов (или панель инструментов диаграммы), необходимо выполнить следующие действия.

1. Выбрать пункт Tools > Options.
2. Выбрать вкладку Toolbars.
3. Чтобы сделать видимой или невидимой стандартную панель инструментов, необходимо пометить (или снять пометку) контрольный переключатель Show Standard ToolBar (или Show Diagram ToolBar).

Чтобы увеличить размер кнопок на панели инструментов, необходимо выполнить следующие действия.

1. Щелкнуть правой кнопкой мыши на требуемой панели.
2. Выбрать во всплывающем меню пункт Use Large Buttons (использовать большие кнопки).

Чтобы настроить панель инструментов, необходимо выполнить следующие действия.

1. Щелкнуть правой кнопкой мыши на требуемой панели.
2. Выбрать пункт Customize (настроить).

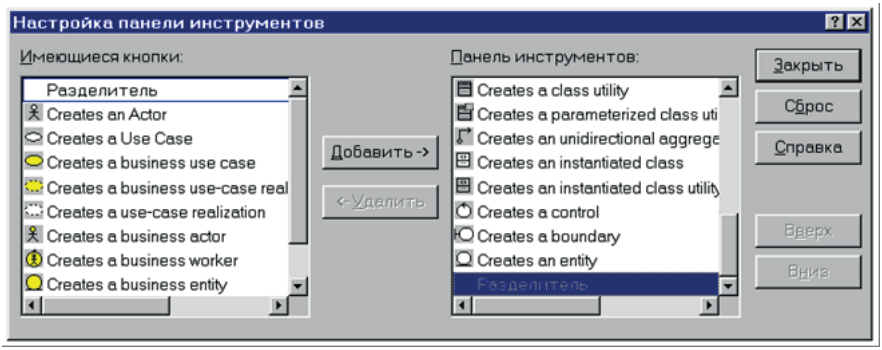


Рис. 4.2. Настройка стандартной панели инструментов

Чтобы добавить или удалить кнопки, необходимо выбрать соответствующую кнопку и затем щелкнуть мышью на кнопке Add (добавить) или Remove (удалить), как это показано на рис. 4.2.

Существует два способа удалить элемент модели — из одной диаграммы или из всей модели. Чтобы удалить элемент модели из диаграммы, необходимо выполнить следующие действия.

1. Выделить элемент на диаграмме.
2. Нажать на клавишу Delete.

Обратите внимания, что, хотя элемент и удален с диаграммы, он остался в браузере и на других диаграммах системы.

Чтобы удалить элемент из модели, необходимо выполнить следующие действия.

1. Выделить элемент на диаграмме.
2. Выбрать пункт меню Edit > Delete from Model или нажать одновременно на клавиши CTRL + D.

Окно диаграммы

В окне диаграммы видно, как выглядит одна или несколько диаграмм UML модели. При внесении в элементы диаграммы изменений Rose автоматически обновит браузер. Аналогично, при внесении изменений в элемент с помощью браузера Rose автоматически

обновит соответствующие диаграммы. Это помогает поддерживать модель в непротиворечивом состоянии.

Журнал

По мере работы над вашей моделью определенная информация будет направляться в окно журнала. Например, туда помещаются сообщения об ошибках, возникающих при генерации кода. Не существует способа закрыть журнал совсем, но его окно может быть минимизировано.

В модели Rose поддерживается четыре представления (views) — представление вариантов использования, логическое представление, представление компонентов и представление размещения. Каждое из них предназначено для своих целей.

Представление вариантов использования

Это представление содержит модель бизнес-процессов и модель вариантов использования. На рисунке 4.1 показано, как выглядит представление вариантов использования в браузере Rose.

Логическое представление

Логическое представление концентрируется на том, как система будет реализовывать поведение, описанное в вариантах использования. Оно дает подробную картину составных частей системы и описывает взаимодействие этих частей. Логическое представление включает в основном классы и диаграммы классов. С их помощью конструируется детальный проект создаваемой системы.

Логическое представление содержит:

– классы;
– диаграммы классов (как правило, для описания системы используется несколько диаграмм классов, каждая из которых отображает некоторое подмножество всех классов системы);

– диаграммы взаимодействия, применяемые для отображения объектов, участвующих в одном потоке событий варианта использования;
– диаграммы состояний;
– пакеты, являющиеся группами взаимосвязанных классов.

Представление компонентов

Представление компонентов содержит:
– компоненты, являющиеся физическими модулями кода;
– диаграммы компонентов;
– пакеты, являющиеся группами связанных компонентов.

Представление размещения

Последнее представление Rose — это представление размещения. Оно соответствует физическому размещению системы, которое может отличаться от ее логической архитектуры.

В представление размещения входят следующие компоненты:
– процессы, являющиеся потоками (threads), исполняемыми в отведенной для них области памяти;
– процессы, включающие любые компьютеры, способные обрабатывать данные. Любой процесс выполняется на одном или нескольких процессорах;
– устройства, то есть любая аппаратура, не способная обрабатывать данные. К числу таких устройств относятся, например, терминалы ввода-вывода и принтеры;
– диаграмма размещения.

Параметры настройки отображения

В Rose имеется возможность настроить диаграммы классов так, чтобы:
– показывать все атрибуты и операции;

– скрыть операции;
– скрыть атрибуты;
– показывать только некоторые атрибуты или операции;
– показывать операции вместе с их полными сигнатурами или только их имена;
– показывать или не показывать видимость атрибутов и операций;
– показывать или не показывать стереотипы атрибутов и операций.

Значения каждого параметра по умолчанию можно задать с помощью окна, открываемого при выборе пункта меню Tools > Options.

У данного класса на диаграмме можно:
– показать все атрибуты;
– скрыть все атрибуты;
– показать только выбранные вами атрибуты;
– подавить вывод атрибутов.

Подавление вывода атрибутов приведет не только к исчезновению атрибутов с диаграммы, но и к удалению линии, показывающей место расположения атрибутов в классе.

Существует два способа изменения параметров представления атрибутов на диаграмме. Можно установить нужные значения у каждого класса индивидуально. Можно также изменить значения нужных параметров по умолчанию до начала создания диаграммы классов. Внесенные таким образом изменения повлияют только на вновь создаваемые диаграммы.

Чтобы показать все атрибуты класса, необходимо выполнить следующие действия.

1. Выделить на диаграмме нужный класс.
2. Щелкнуть на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В контекстно-зависимом меню нужно выбрать Options > Show All Attributes.

Чтобы показать у класса только избранные атрибуты, необходимо выполнить следующие действия.

1. Выделить на диаграмме нужный вам класс.

2. Щелкнуть на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем нужно выбрать Options > Select Compartment Items.
4. Указать нужные атрибуты в окне Edit Compartment.

Чтобы подавить вывод всех атрибутов класса диаграммы, необходимо выполнить следующие действия.

1. Выделить на диаграмме нужный класс.
2. Щелкнуть на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем нужно выбрать Options > Suppress Attributes.

Чтобы изменить принятый по умолчанию вид атрибута, необходимо выполнить следующие действия.

1. В меню модели выберать пункт Tools > Options.
2. Перейти на вкладку Diagram.
3. Для установки значений параметров отображения атрибутов по умолчанию нужно воспользоваться контрольными переключателями Suppress Attributes и Show All Attributes. Изменение этих значений по умолчанию повлияет только на новые диаграммы. Вид существующих диаграмм классов не изменится.

Как и в случае атрибутов, имеется несколько вариантов представления операций на диаграммах:

– показать все операции;
– показать только некоторые операции;
– скрыть все операции;
– подавить вывод операций.

Кроме того, можно выбрать следующие ниже варианты:

– показать только имя операции. Это означает, что на диаграмме будет представлено только имя операции, но не аргументы или тип возвращаемого значения;
– показать полную сигнатуру операции. На диаграмме будет представлено не только имя операции, но и все ее параметры, типы данных параметров и тип возвращаемого значения операции.

Чтобы показать все операции класса, необходимо выполнить следующие действия.

1. Выделить на диаграмме нужный вам класс.
2. Щелкнуть на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В контекстно-зависимом меню выбрать Options > Show All Operations.

Чтобы показать только избранные операции класса, необходимо выполнить следующие действия.

1. Выделить на диаграмме нужный вам класс.
2. Щелкнуть на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выбрать Options > Select Compartment Items.
4. Указать нужные операции в окне Edit Compartment.

Чтобы подавить вывод всех операций класса диаграммы, необходимо выполнить следующие действия:

1. Выделить на диаграмме нужный вам класс.
2. Щелкнуть на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выбрать Options > Suppress Operations.

Чтобы показать на диаграмме классов сигнатуру операции, необходимо выполнить следующие действия.

1. Выделить на диаграмме нужный вам класс.
2. Щелкнуть на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выбрать Options > Show Operation Signature.

Чтобы изменить принятый по умолчанию вид операции, необходимо выполнить следующие действия.

1. В меню модели нужно выбрать пункт Tools > Options.
2. Затем перейти на вкладку Diagram.
3. Для установки значений параметров отображения операций по умолчанию нужно воспользоваться контрольными переключателями Suppress Operations, Show All Operations и Show Operation Signatures.

Чтобы показать видимость атрибута или операции класса, необходимо выполнить следующие действия.

1. Выделить на диаграмме нужный класс.
2. Щелкнуть на нем правой кнопкой мыши, чтобы открыть контекстно-зависимое меню.
3. В нем выбрать Options > Show Visibility.

Чтобы изменить принятое по умолчанию значение параметра показа видимости, необходимо выполнить следующие действия.

1. В меню модели выбрать пункт Tools > Options.
2. Перейти на вкладку Diagram.
3. Для установки параметров отображения видимости по умолчанию нужно воспользоваться контрольным переключателем Show Visibility.

Для переключения между нотациями видимости Rose и UML необходимо выполнить следующие действия.

1. В меню модели выберать пункт Tools > Options.
2. Перейти на вкладку Notation.
3. Для переключения между нотациями нужно воспользоваться переключателем Visibility as Icons. Если этот переключатель помечен, будет использоваться нотация Rose. Если нет, то нотация UML. Изменение этого параметра повлияет только на новые диаграммы.

Существующие диаграммы классов останутся прежними.

Содержание отчета о выполненной работе

В отчет следует включить следующие пункты.

1. Цель работы
2. Введение.
3. Краткое описание целей проекта и проектных ограничений (бюджетных, временных и т.д.), которые важны для управления проектом.
4. Описание информационной системы (ПО) — наличие заключения о возможности реализации проекта, содержащего реко-

мендации относительно разработки системы, базовые предложения по объёму требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению.

5. Анализ осуществимости (согласно требованиям к результатам выполнения лабораторного практикума п. 2) — указать возможные проблемы и пути их решения.
6. Роли участников группы разработки ПО.
7. Программно-аппаратные средства, используемые при выполнении работы.
8. Выводы.

5. ЗАДАНИЯ К ПРАКТИЧЕСКИМ РАБОТАМ

Содержание отчета о практической работе

В отчетах по практическим работам следует представить:

1) цель работы;
2) краткое описание целей проекта и проектных ограничений (бюджетных, временных и т.д.), которые важны для управления проектом;
3) описание информационной системы (ПО) — наличие заключения о возможности реализации проекта, содержащего рекомендации относительно разработки системы, базовые предложения по объёму требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению;
4) анализ осуществимости (согласно требованиям к результатам выполнения лабораторного практикума п. 2), указать возможные проблемы и пути их решения;
5) роли участников группы разработки ПО;
6) программно-аппаратные средства, используемые при выполнении работы;
7) заключение (выводы).

ПРАКТИЧЕСКАЯ РАБОТА № 1
СОСТАВЛЕНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ НА РАЗРАБОТКУ ИС

Цель работы

Провести анализ предметной области и составить техническое задание на разработку ИС.

Общие сведения

Задание к лабораторной работе

1. Изучить предлагаемый теоретический материал.
2. Составить подробное описание проектируемой информационной системы.
3. На основании описания системы провести анализ осуществимости. Результатом анализа должно явиться заключение о возможности реализации проекта.
4. Распределить роли в группе (руководитель проекта-разработчик, системный аналитик-разработчик, тестер-разработчик).
5. Заполнить следующие разделы плана.
   1) Введение.
   2) Организация выполнения проекта.
   3) Анализ рисков.

   Разделы должны содержать рекомендации относительно разработки системы, базовые предложения по объёму требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению.
6. Составить отчет о проделанной работе.

Контрольные вопросы

1. Что произойдет с организацией, если система не будет введена в эксплуатацию?
2. Какие текущие проблемы существуют в организации и как новая ИС поможет их решить?
3. Каким образом система будет способствовать целям бизнеса?
4. Требует ли разработка системы технологии, которая до этого не использовалась в организации?

ПРАКТИЧЕСКАЯ РАБОТА № 2
ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Цель работы

Провести объектно-ориентированный анализ предметной области и построить концептуальную диаграмму классов.

Общие сведения

Анализ состоит в исследовании проблемы, а не в поисках путей ее решения. В процессе анализа осуществляется декомпозиция сложной проблемы на более мелкие составные части. При структурном анализе такими частями являются функции системы (процедуры), при объектно-ориентированном анализе система представляется в виде взаимодействующих объектов.

В ходе объектно-ориентированного анализа необходимо идентифицировать основные понятия, атрибуты и ассоциации из предметной области, имеющие существенное значение для решения задачи. Результат анализа выражается в модели предметной области, которую можно представить в виде диаграммы классов языка UML. Следует помнить, что модель предметной области — это не описание программных компонентов. Это представление понятий, выраженных в терминах предметной области задачи.

На диаграммах UML классы обозначаются в виде прямоугольника. Внутри прямоугольника записывается имя класса. По традиции имя класса пишется с большой буквы. Внутри прямоугольника выделены секция атрибутов и секция операций. Примеры обозначения классов есть на рис. 5.1.

Рис. 5.1. Обозначение класса:
а) без дополнительных секций;
б) с секциями атрибутов и операций

Идентификацию концептуальных классов удобно начинать с анализа текстового описания предметной области. Выделенные в тек-

сте существительные рассматриваются в качестве кандидатов в концептуальные классы или атрибуты.

Рассмотрим в качестве примера станцию технического обслуживания (СТО). Сначала составим неформальное текстовое описание бизнес-процессов этой организации.

Клиент (владелец автомобиля) приезжает на станцию техобслуживания (СТО). Мастер-приемщик принимает автомобиль у клиента и выполняет его внешний осмотр. Механик выполняет техническое обслуживание (ТО) в соответствии с перечнем операций, входящих в ТО. Клиент оплачивает ТО через кассу. Мастер-приемщик делает запись в книге ТО автомобиля и выдает автомобиль клиенту. Мастер-приемщик, механик и кассир являются сотрудниками СТО.

В этом текстовом описании можно выделить следующий список кандидатур на роль концептуальных классов:

– автомобиль;
– техническое обслуживание;
– сотрудник;
– механик;
– мастер-приемщик;
– кассир;
– платеж;
– книга ТО;
– запись о проведении ТО.

Другой способ выявления концептуальных классов состоит в поиске понятий предметной области, попадающих в одну из стандартных категорий.

В данном случае можно составить следующий список понятий:

– физические или материальные объекты (автомобиль, запасная часть);
– спецификации или описания объектов (описание услуги, описание товара);
– транзакции (продажа, резервирование, платеж);
– роли людей (механик, бухгалтер, мастер-приемщик);
– контейнеры других объектов (автосалон);

– содержимое контейнеров (автомобиль);
– другие системы, внешние по отношению к данной системе (система заказа запасных частей);
– абстрактные понятия (качество, надежность);
– организации (отдел продаж, механический участок);
– события (продажа, платеж, гарантийный ремонт);
– правила и политика (правила проведения технического облуживания);
– каталоги (каталог автомобилей, каталог запасных частей);
– записи финансовой, трудовой, юридической и другой деятельности (трудовой контракт, договор, журнал обслуживания);
– финансовые инструменты и службы (кредитная линия, налоговая инспекция);
– руководства, документы, статьи, книги (руководство по ремонту).

Следующим шагом при построении модели предметной области является определение ассоциаций между классами.

Ассоциация — это связь между классами (или, точнее, экземплярами классов), отражающая некоторое значимое и полезное отношение между ними. На стадии создания модели предметной области ассоциация не является описанием потоков данных, экземпляров переменных или взаимодействия объектов программной системы. Она представляет собой лишь описание значимого отношения, имеющего чисто концептуальный смысл, почерпнутый из реального мира.

В общем случае ассоциация может объединять несколько классов, но чаще всего встречаются бинарные ассоциации, объединяющие два класса.

На диаграммах UML ассоциация изображается в виде линии, соединяющей два класса. Имя ассоциации записывается над линией, как показано на рис. 5.2.

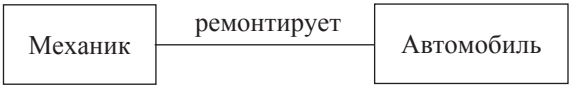


Рис. 5.2. Ассоциация между классами

При правильном построении модели предметной области имена классов и имя ассоциации формируют грамматически правильное предложение. Например, ассоциацию, показанную на рис. 5.2, можно прочитать так: «Механик ремонтирует автомобиль». Дополнительная стрелка рядом с именем ассоциации указывает, в каком направлении нужно читать ее имя. Она не определяет направление видимости или перемещения. Если такая стрелка отсутствует, то имена ассоциаций следует читать с использованием общепринятых соглашений, а именно — слева направо и сверху вниз.

Каждый класс в ассоциации играет определенную роль. Роль класса записывается рядом с соответствующим окончанием линии, связывающей классы. Для класса, участвующего в ассоциации, может быть также определена кратность: количество экземпляров данного класса, которые могут входить в ассоциацию. Пример обозначения ролей и кратности показан на рис. 5.3.

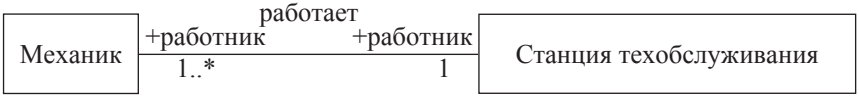


Рис. 5.3. Роли классов и кратность

Как видно из рис. 5.3, в данной ассоциации класс «Механик» играет роль работника, а класс «Станция техобслуживания» — роль работодателя. Из рисунка также видно, что несколько механиков могут работать на одной станции техобслуживания.

Некоторые разновидности ассоциаций имеют специальное обозначение. Отношение агрегации — это отношение «часть – целое». Пример обозначения отношения агрегации на диаграммах UML показан на рис. 5.4.

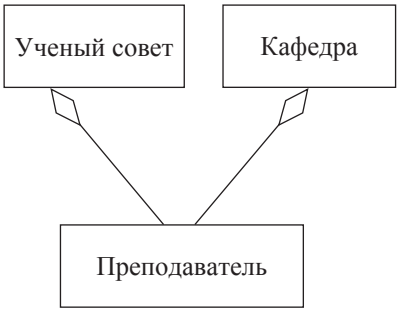


Рис. 5.4. Отношение агрегации

Отношения агрегации, показанные на рисунке, читаются как:

«Преподаватель входит в состав ученого совета»; «Преподаватель входит в состав кафедры». Отношение агрегации допускает, что один объект входит в состав нескольких объектов.

Отношение композиции — это тоже отношение «часть – целое», но более сильное, чем агрегация. Объект может быть частью только одного целого.

Пример обозначения отношения композиции на диаграммах UML показан на рис. 5.5.

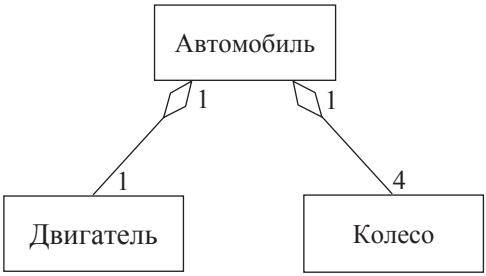


Рис. 5.5. Отношение композиции

Отношение композиции можно прочесть так: «Двигатель является частью автомобиля»; «Колесо является частью автомобиля». Кратность отношения показывает, что один автомобиль имеет четыре колеса и один двигатель.

Отношение обобщения используется в диаграммах UML для обозначения наследования. Пример обозначения отношения обобщения показан на рис. 5.6.

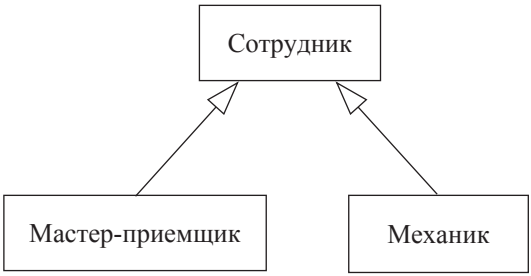


Рис. 5.6. Отношение обобщения

Отношение обобщения можно прочесть так: «Мастер-приемщик является разновидностью сотрудника»; «Механик является разновидностью сотрудника».

При поиске ассоциаций между классами основное внимание нужно уделить тем ассоциациям, знания о которых нужно сохранять в течение некоторого периода (важным ассоциациям).

Слишком большое количество ассоциаций приводит к ошибкам в модели предметной области, а не к ее упрощению. Следует избегать отображения избыточных или не имеющих самостоятельного значения ассоциаций.

При поиске ассоциаций полезно держать в памяти, следующие стандартные категории ассоциаций между классами «A» и «B»:

– «A» является физической частью «B»;
– «A» является логической частью «B»;
– «A» физически содержится в/на «B»;
– «A» логически содержится в «B»;
– «A» является описанием «B»;
– «A» является элементом транзакции или отчета «B»;
– «A» известен/зарегистрирован/записан/включен в «B»;
– «A» является членом «B»;
– «A» является организационной единицей «B»;
– «A» использует или управляет «B»;
– «A» взаимодействует с «B»;
– «A» связан с транзакцией «B»;
– «A» является транзакцией, которая связана с другой транзакцией «B»;
– «A» следует за «B»;
– «A» является «собственностью» «B»;
– «A» является событием, связанным с «B».

На примере станции технического обслуживания можно выделить следующие ассоциации:

– мастер-приемщик является сотрудником СТО;
– кассир является сотрудником СТО;
– механик является сотрудником СТО;
– клиент владеет автомобилем;
– мастер-приемщик принимает автомобиль;
– механик выполняет техническое обслуживание (ТО);
– клиент осуществляет платеж;
– кассир принимает платеж;
– мастер-приемщик делает запись о проведении ТО;

– запись о проведении ТО является частью книги ТО;

– мастер-приемщик выдает автомобиль.

Итоговая модель предметной области станции технического обслуживания показана на рис. 5.7.

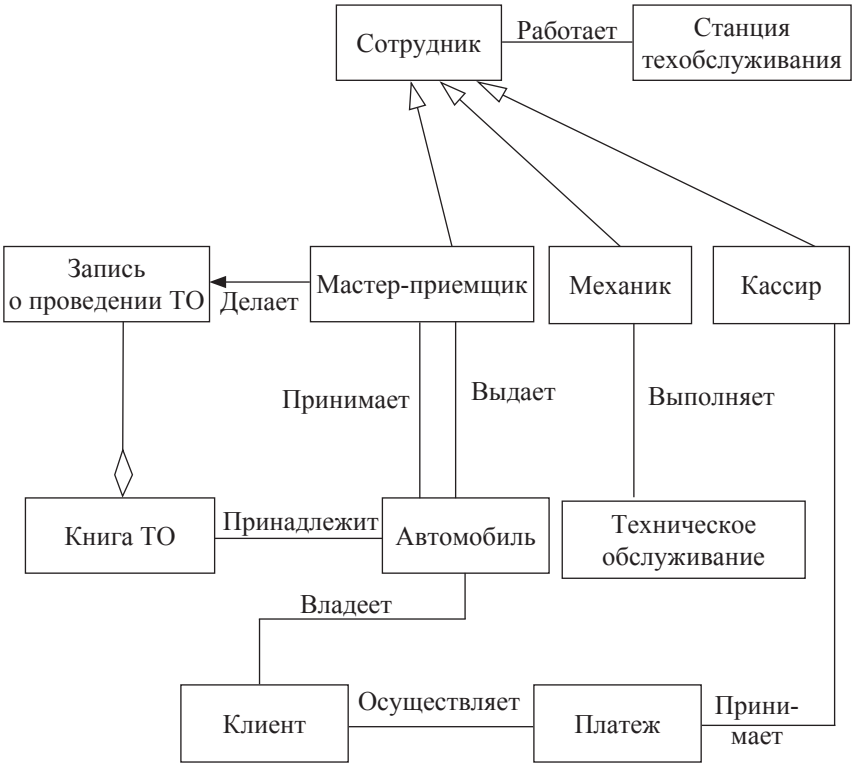


Рис. 5.7. Модель предметной области СТО

Задание к лабораторной работе

1. Изучить предлагаемый теоретический материал.
2. Составить текстовое описание предметной области, указанной преподавателем.
3. Согласовать текстовое описание с преподавателем.

4. Используя текстовое описание, идентифицировать основные классы предметной области.
5. Идентифицировать ассоциации между классами.
6. С помощью программы Rational Rose создать модель предметной области.
7. Сделать выводы по работе.

Контрольные вопросы

1. В чем состоит цель объектно-ориентированного анализа?
2. Что такое модель предметной области?
3. Какие методы идентификации концептуальных классов вы знаете?
4. Для каких типов ассоциаций в UML есть специальные обозначения?
5. Как можно идентифицировать ассоциации между классами?

ПРАКТИЧЕСКАЯ РАБОТА № 3
РАЗРАБОТКА ТРЕБОВАНИЙ К СИСТЕМЕ

Цель работы

Познакомиться с методикой описания требований к системе в форме прецедентов. Выделить основные прецеденты разрабатываемой системы, выполнить текстовое описание прецедентов и построить диаграмму прецедентов.

Общие сведения

Требования — это возможности или условия, которым должна соответствовать система или проект. Основная задача этапа определения требований — найти, обсудить и зафиксировать, что действительно требуется от системы в форме, понятной и клиентам, и членам команды разработчиков.

Требования можно разделить на две большие категории: функциональные (относящиеся к поведению) и нефункциональные: добство, надежность, производительность, возможность поддержки и другие.

Функциональные требования исследуются и формулируются в процессе разработки модели прецедентов. Остальные требования формулируются при более детальном описании прецедентов или в дополнительной спецификации.

У потребителей и конечных пользователей компьютерной системы есть свои задачи, решение которых должна обеспечить компьютерная система. Существует несколько способов выделения этих задач или системных требований. Наилучшие из них достаточно просты и доступны, поскольку это облегчает участие конечных пользователей в определении требований к системе.

Исполнителем называют сущность, обладающую поведением, например, человека, компьютерную систему или организацию.

Сценарий — последовательность действий или взаимодействий между исполнителем и системой. Это один конкретный сценарий использования системы, например, сценарий успешной покупки товара за наличный расчет либо сценарий неудачного завершения

покупки из-за прерванной транзакции по обработке кредитной карточки.

Прецедент — набор взаимосвязанных успешных и неудачных сценариев, описывающих использование системы исполнителем для решения одной из своих задач.

Выделение прецедента. Каждую задачу можно рассматривать на разных уровнях детализации, начиная от конкретных простых действий и заканчивая деятельностью на уровне предприятия. В процессе анализа требований к компьютерному приложению следует сосредоточить внимание на уровне элементарных бизнес-процессов.

Элементарный бизнес-процесс — это задача, выполняемая одним человеком в одном месте и в одно время в ответ на некоторое бизнес-событие и переводящая данные в некоторое устойчивое состояние.

Это определение можно воспринимать слишком буквально. Тогда возникает вопрос: могут ли в сценарии прецедента участвовать два человека? Скорее всего, да. Однако общая идея этого определения верна. Прецедент — это не один маленький шаг, такой, например, как удаление товара или печать документа. Основной сценарий прецедента обычно включает пять-десять шагов. Описываемый сценарий выполняется не в течение нескольких дней или сеансов, как, например, переговоры с поставщиками. Это задача, выполняемая в течение одного сеанса. Реализация прецедента может длиться от нескольких минут до нескольких дней.

Несмотря на то, что основные прецеденты приложения должны соответствовать элементарным бизнес-процессам, зачастую полезно создавать отдельные прецеденты более низкого уровня, представляющие подзадачи в рамках основного прецедента. Например, подзадача «Оплата по кредитной карточке» может быть представлена в нескольких основных прецедентах. Поэтому ее желательно выделить в отдельный прецедент (хотя он и не соответствует условиям элементарного бизнес-процесса) и связать с несколькими основными прецедентами, чтобы избежать дублирования информации.

Прецеденты предназначены для удовлетворения потребностей основных исполнителей. Поэтому для выделения прецедентов используется следующая процедура.

1. Определите рамки системы: является ли она программным приложением, аппаратно-программным комплексом, включает ли в себя своих пользователей или всю организацию?
2. Идентифицируйте основных исполнителей, потребности (цели) которых удовлетворяются с помощью системы.
3. Для каждого исполнителя определите его задачи.
4. Определите прецеденты, удовлетворяющие потребности каждого исполнителя, и присвойте им имена в соответствии с задачами.

Как правило, каждой задаче пользователя соответствует один прецедент. Его имя должно соответствовать названию задачи, например, задаче оформления продажи должен соответствовать прецедент «Оформление продажи». Типичным исключением из правила соответствия задач и прецедентов является прецедент, решающий четыре задачи — создание, восстановление, обновление и удаление. Обычно такой прецедент называется «Управление чем-либо». Например, задачи «изменение информации о пользователях», «удаление пользователей» и т.д. решаются в рамках прецедента «Управление пользователями».

Рассматривая систему автоматизации бизнес-процессов станции технического обслуживания автомобилей, можно выделить следующих основных исполнителей:

- мастер-приемщик;
- система заказа запчастей (внешняя система).

Для мастера-приемщика можно выделить следующие задачи:

- прием автомобиля;
- ремонт или техническое обслуживание автомобиля;
- выдача автомобиля.

Прецеденты могут записываться с различной степенью детализации. Первоначально имеет смысл описать прецеденты в сжатой форме, в виде неформального рассказа «из жизни системы».

Опишем перечисленные выше прецеденты в сжатой форме.

Приемка автомобиля. Для прохождения ТО владелец автомобиля (клиент) приезжает на станцию технического обслуживания. Мастер-приемщик составляет заявку на ремонт автомобиля, в которой фиксируется модель автомобиля, VIN, год выпуска, пробег, а также неисправности автомобиля со слов клиента. Далее мастер-приемщик выполняет наружный осмотр автомобиля и делает соответствующие пометки в заявке на ремонт. Клиент расписывается в заявке на ремонт.

Техническое обслуживание автомобиля. Автомобиль загоняется на механический участок, где выполняются операции, предусмотренные регламентом технического обслуживания. Необходимый ремонт или замена деталей выполняется после предварительно согласования с клиентом. Замененные детали передаются клиенту. Мастер-приемщик делает запись о прохождении ТО в книге технического обслуживания автомобиля.

Выдача автомобиля. Стоимость выполненных работ, запасных частей и материалов вносится в заказ-наряд. Клиент осуществляет платеж. Мастер-приемщик передает автомобиль клиенту.

Пример развернутого описания прецедента приведен в табл. 5.1.

Таблица 5.1

Прецедент «Приемка автомобиля»

Основной исполнитель: мастер-приемщик
Заинтересованные лица и их требования 1. Мастер-приемщик. Хочет быстро выполнить прием автомобиля, без ошибок зафиксировать сведения об автомобиле, его состоянии, идентифицировать неисправности автомобиля. 2. Владелец автомобиля. Хочет быстро оформить прием автомобиля, убедиться, что информация о неисправностях автомобиля правильно понята и зафиксирована. 3. Руководство станции технического обслуживания. Хочет удовлетворить интересы владельца автомобиля, правильно зафиксировать информацию об автомобиле.

4. Фирма-производитель автомобиля. Хочет контролировать качество сервисного обслуживания автомобилей, получить точную информацию о неисправностях автомобиля
Основной успешный сценарий
1. Владелец автомобиля передает мастеру-приемщику свидетельство о регистрации транспортного средства. 2. Мастер-приемщик вводит в систему информацию об автомобиле: регистрационный знак, VIN, марку и модель автомобиля, номер двигателя, пробег автомобиля, неисправности автомобиля со слов клиента, а также информацию о владельце: фамилию, имя и отчество. 3. Система распечатывает заявку на ремонт автомобиля. 4. Мастер-приемщик выполняет наружный осмотр автомобиля и делает пометки в заявке на ремонт автомобиля. 5. Мастер-приемщик беседует с владельцем автомобиля для более точной идентификации неисправностей. 6. Клиент расписывается в заявке на ремонт автомобиля и забирает себе один из двух экземпляров заявки.
Расширения: 2а. Автомобиль известен
1. Если введенный регистрационный знак автомобиля уже известен системе, система автоматически заполняет VIN, марку и модель автомобиля, номер двигателя и информацию о владельце. 2. Мастер-приемщик проверяет корректность этой информации, сравнивая ее со свидетельством о регистрации транспортного средства.

Диаграмма прецедентов языка UML отображает зависимости между прецедентами и исполнителями. Диаграммы прецедентов используются при работе над прецедентами, но играют при этом вспомогательную роль и дополняют текстовые описания возможных вариантов использования разрабатываемой системы.

Прецеденты отображаются в текстовых документах. Работать над прецедентами — значит составлять текстовые описания. В то же время диаграмма прецедентов — это отличное изображение системного контекста, поскольку она отображает границы системы, внешние для системы понятия и способы использования системы.

Диаграмма прецедентов для станции технического обслуживания показана на рис. 5.8.

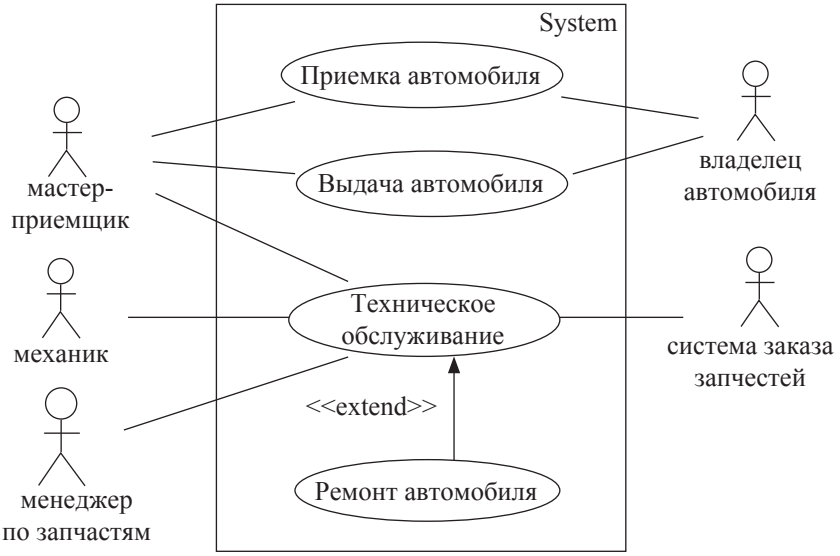


Рис. 5.8. Пример диаграммы прецедентов

Задание к лабораторной работе

1. Изучить предлагаемый теоретический материал.
2. Определить рамки системы: является ли она программным приложением, аппаратно-программным комплексом, включает ли в себя своих пользователей или всю организацию.
3. Идентифицировать основных исполнителей, потребности (цели) которых удовлетворяются с помощью системы.
4. Для каждого исполнителя определить его задачи.
5. Определить прецеденты, удовлетворяющие потребности каждого исполнителя, и присвоить им имена в соответствии с задачами.
6. Описать все прецеденты в сжатом виде.
7. Выполнить описание одного из прецедентов (по выбору преподавателя) в развернутом виде.
8. С помощью программы Rational Rose создать диаграмму прецедентов.
9. Сделать выводы по работе.

Контрольные вопросы

1. Какие категории требований к системе вы знаете?
2. Что подразумевается под прецедентом в языке UML?
3. Какая категория требований может быть описана с помощью прецедентов?
4. Каким образом можно выделить прецеденты?
5. Какие способы описания прецедентов вы знаете?
6. Какие обозначения используются на диаграмме прецедентов?

ПРАКТИЧЕСКАЯ РАБОТА № 4
ВИЗУАЛИЗАЦИЯ БИЗНЕС-ПРОЦЕССОВ

Цель работы

Построить диаграмму деятельности, описывающую бизнес-процесс.

Общие сведения

Для описания алгоритма выполнения прецедента составляют диаграммы кооперацій — диаграммы взаимодействия или диаграммы последовательностей. Построение диаграммы кооперации для прецедента происходит на основе построенной диаграммы классов — участников прецедента. Если в прецеденте существует несколько возможных потоков управления, то составляют по одной диаграмме кооперации на каждый поток.

На диаграмму кооперации помещают объекты классов, участвующих в ней, и добавляют сообщения, которыми обмениваются объекты при реализации прецедента. Линии и направления сообщений соответствуют ассоциациям на диаграмме классов участников.

Последовательность сообщений определяется порядком взаимодействия объектов. Начинается кооперация всегда сообщением от актера-инициатора, то есть того участника отношений, который начинает взаимодействие. Сообщения указываются без сигнатур, только названием ответственности.

Диаграммы деятельности являются средством описания поведения в UML, причем их место в языке допускает некоторые разночтения.

Основных сущностей и отношений, применяемых на диаграмме деятельности, в некотором смысле еще меньше, чем на диаграмме состояний (хотя, казалось бы, меньше уже некуда — на диаграмме состояний только состояния и переходы). Дело в том, что основная сущность на диаграмме деятельности является частным случаем простого состояния (состояние деятельности), а основное отношение — частным случаем простого перехода (переход по завершении). В то же время всевозможных украшений и вариантов нотации на диаграмме деятельности, особенно в UML 2, намного больше, чем на диаграмме

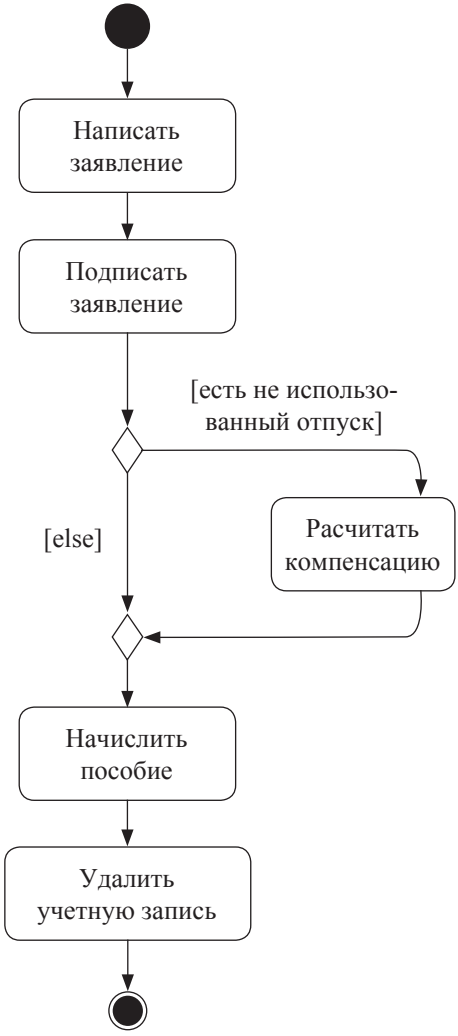


Рис. 5.9. Диаграмма деятельности

автомата. Поэтому, чтобы не затеряться в деталях, в следующем параграфе мы обсудим содержание базовых понятий, затем определим основные сущности и отношения, применяемые на диаграммах деятельности, а уже потом перейдем к примерам и картинкам.

Действия используются на переходах и в состояниях UML и играют там ключевую роль. Каждое действие имеет присущие ему наборы входных и выходных параметров, которые называются контактами (pin). Как правило, эти наборы фиксированы по числу и типам параметров, но бывают и действия с изменяемым числом параметров. Среди элементарных действий нет привычных действий по управлению ходом выполнения программы (ветвления, циклы, переходы и т.д.) — управление не считается примитивом и вынесено на следующий уровень, уровень деятельности.

Вторым важнейшим понятием, применяемым при описании поведения, является деятельность. Деятельность (activity) в UML — это описание поведения в форме графа деятельности. Дея-

тельность в UML моделирует то же, что и действие, т.е. какую-то содержательную активность во время работы системы; в этом смысле деятельность подобна действию, но деятельность противопоставляется действию по всем характеристическим признакам.

Если нам не важно различие между действием и деятельностью и требуется употребить более общее понятие, то применяется термин «активность».

Пример диаграммы деятельности представлен на рис. 5.9.

На диаграмме деятельности UML применяется один основной тип отношений — простые переходы по завершении (а также поток объектов). Переход по завершении не имеет переключающего события — событием является завершение внутренней активности (деятельности) в состоянии. Как правило, исходящий переход по завершении один; если их несколько, они должны быть снабжены сторожевыми условиями, образующими полную дизъюнктную систему предикатов. Кроме того, в UML можно использовать и переходы, возбуждаемые событиями. Срабатывание такого перехода означает прерывание выполнения деятельности в состоянии и переход в другое состояние. Однако использование таких переходов неуместно на диаграмме деятельности. Пример диаграммы деятельности СТО для функционального описания задач, рассмотренных выше, приведен на рис. 5.10.

Задание к лабораторной работе

1. Изучить предлагаемый теоретический материал.
2. Для указанного преподавателем бизнес-процесса составить неформальное текстовое описание.
3. Выделить действия, которые могут выполняться параллельно.
4. Определить зоны ответственности (дорожки).
5. С помощью программы Rational Rose нарисовать диаграмму деятельности.
6. Сделать выводы по работе.

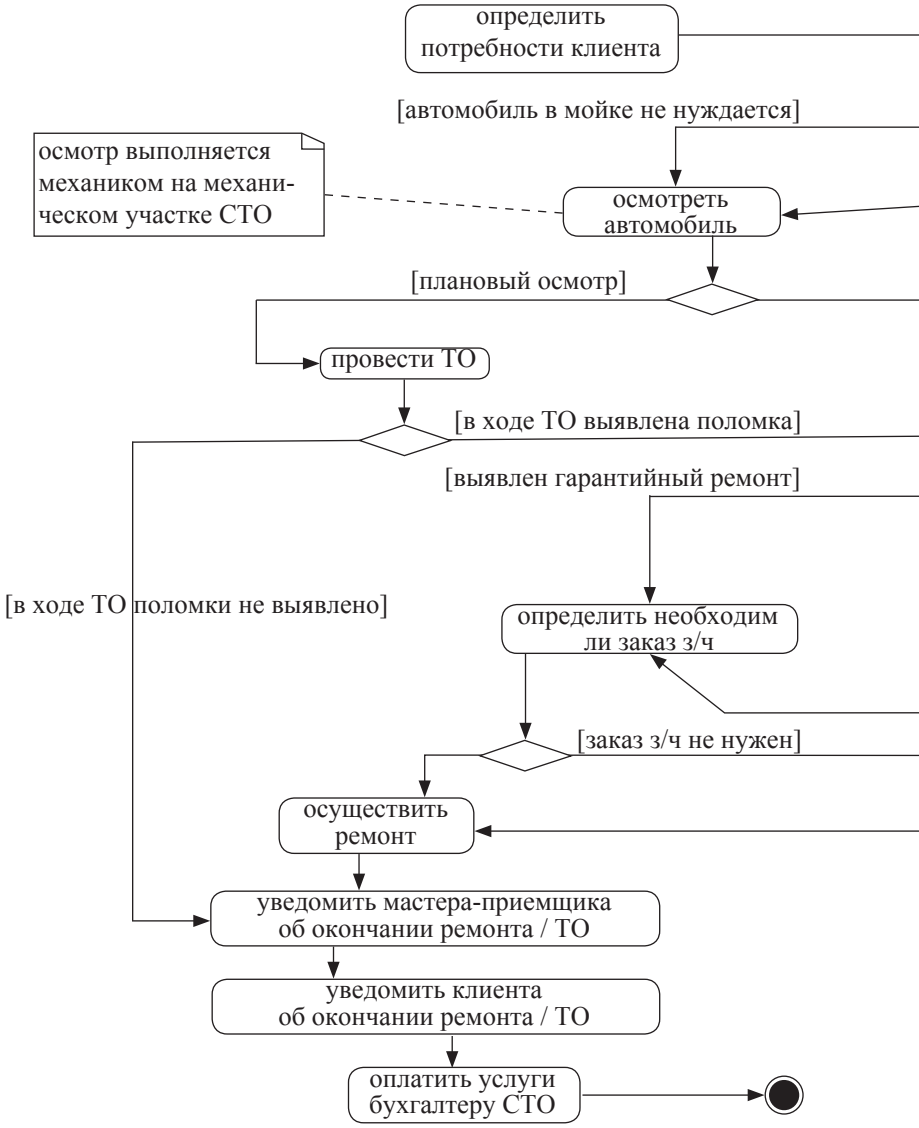
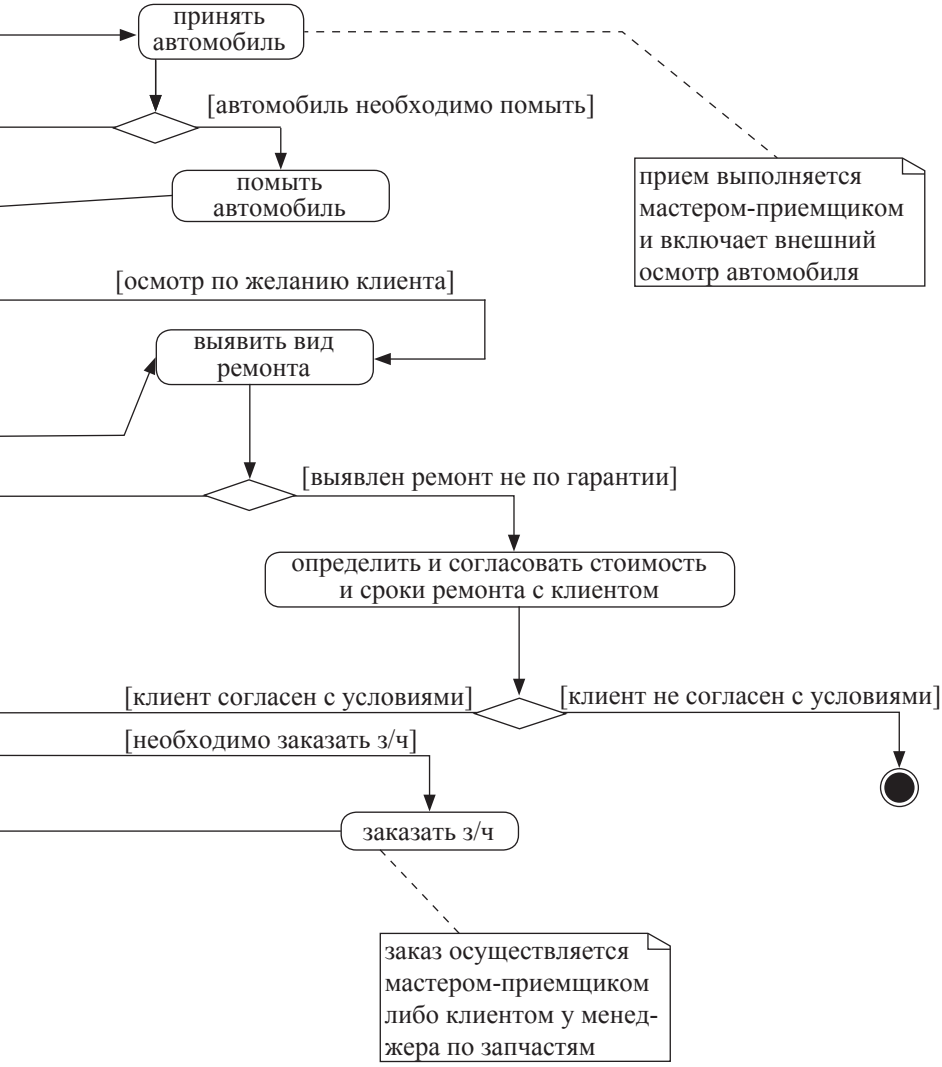


Рис. 5.10. Пример диаграммы деятельности СТО


принять автомобиль
[автомобиль необходимо помыть]
помыть автомобиль
прием выполняется мастером-приемщиком и включает внешний осмотр автомобиля
[осмотр по желанию клиента]
выявить вид ремонта
[выявлен ремонт не по гарантии]
определить и согласовать стоимость и сроки ремонта с клиентом
[клиент согласен с условиями]
[клиент не согласен с условиями]
[необходимо заказать з/ч]
заказать з/ч
заказ осуществляется мастером-приемщиком либо клиентом у менеджера по запчастям

Контрольные вопросы

1. В чем сходство между диаграммой состояний и диаграммой деятельности?
2. Каким образом на диаграмме деятельности моделируются параллельные процессы?
3. Каким образом на диаграмме деятельности моделируются ветвления по условию?
4. Какие обозначения используются на диаграмме деятельности?

ПРИЛОЖЕНИЯ

Приложение 1

Приемы формирования требований

Сбор информации	Анализ	Спецификации	Проверка
• Определите компетенцию продукта и границы проекта	• Сформулируйте среду приложения	• Используйте шаблон спецификации требований	• Изучите документы с требованиями
• Определите классы пользователей	• Создайте прототипы	• Определите источники требований	• Протестируйте тестирования
• Выделите из пользователей ярых сторонников продукта	• Проанализируйте осуществимость	• Задайте каждому требованию уникальные идентификаторы	• Определите критерии приемлемости
• Создайте фокус-группы	• Расставьте приоритеты для требований	• Задокументируйте бизнес-правила	• Смоделируйте требования
• Определите пользовательские требования	• Создайте словарь данных	• Определите атрибуты качества	
• Определите системные события и реакцию на них	• Смоделируйте требования		
• Проводите интервью для выявления требований	• Проанализируйте интерфейсы		

Продолжение табл. 1

Сбор информации	Анализ	Спецификации	Проверка
• Проводите семинары по выявлению требований	• Распределите требования по подсистемам		
• Наблюдайте за пользователями на рабочих местах			
• Раздайте опросные листы			
• Выполните анализ документов			
• Изучите отчеты о проблемах			
• Повторно задействуйте существующие требования			

Приемы формирования требований

Управление требованиями	Обучение	Управление проектом
• Определите процесс управления изменениями	• Обучите аналитиков требований	• Выберите соответствующий цикл разработки проекта
• Проанализируйте, какое влияние оказывают изменения	• Ознакомьте представителей пользователей и менеджеров с требованиями	• Планируйте подход к работе с требованиями

• Определите базовую и контрольную версии наборов требований	• Обучите разработчиков основам предметной области	• Оцените объем работ по реализации требований
• Отслеживайте хронологию изменений	• Определите процесс разработки требований	• Планируйте на основе требований
• Отслеживайте состояние требований	• Создайте словарь терминов	• Определите лиц, ответственных за принятие решений по требованиям
• Отслеживайте проблемы с требованиями		• Своевременно пересматривайте обязательства
• Создайте матрицу связей требований		• Управляйте рисками, касающимися требований
• Используйте средство управления требованиями		• Отслеживайте объем работ по реализации требований
		• Делайте выводы из полученного опыта

Приложение 2

Пример разработанного технического задания на проектирование ИС

УТВЕРЖДАЮ
Учредитель ИП «ГорАвто» (должность)
ФИО
«__»_____2019 г.

Система бронирования проката автомобилей

(*наименование вида системы интернет-заказов*)

СИБ «Прокат автомобилей»

(*сокращенное наименование системы интернет бронирования*)

Действует с _____

СОГЛАСОВАНО
Учредитель «ГорАвто» (должность)
ФИО
«___»_____2019 г.

РАЗРАБОТЧИК
Программист (должность)
ФИО
«___»_____2019 г.

Москва, 2019

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1. Термины, используемые в техническом задании

HTML (HyperText Markup Language — язык гипертекстовой разметки) — это стандартный язык разметки для создания веб-страниц и веб-приложений. **CSS**

CSS (Cascading Style Sheets — каскадные таблицы стилей) используются для описания внешнего вида документа, написанного языком разметки HTML.

JavaScript — мультипарадигменный язык программирования.

SQL (Structured Query Language) — стандартизированный язык программирования, используемый для управления реляционными базами данных и выполнения различных операций с данными в них.

PHP (Hypertext Preprocessor) — распространенный язык программирования общего назначения с открытым исходным кодом. PHP специально сконструирован для веб-разработок, и его код может внедряться непосредственно в HTML.

2. Общие положения

2.1. Название

«Проектирование Web-приложения системы бронирования для проката автомобилей»

2.2. Разработчики и заказчики

Разработчики: ФИО.

Заказчик: ИП «ГорАвто»

2.3. Перечень документов, на основании которых создается сайт

Техническое задание.

План работ.

Акт приема работ.

Акт сдачи в установленный срок.

2.4. Состав и содержание работ по созданию системы

1. Формулировка требований, проектирование, анализ предметной области, составление технического задания.
2. Разработка дизайна.
3. HTML верстка.
4. PHP прописать сценарии сайта.
5. Программирование web-приложения.
6. Наполнение сайта.
7. Тестирование.

3. Назначение и цели создания системы бронирования

Цели

Создание системы бронирования для проката автомобилей.

Задачи

- Загрузка новых материалов, сведений и автомобильного ассортимента.
- Просмотр материалов, сведений и автомобильного ассортимента.
- Добавление материалов, сведений и автомобильного ассортимента.
- Просмотр вспомогательной информации, такой как, например, контакты, инструкции менеджеры, описание материалов и т.д.

4. Требования к сайту и к ПО

4.1. Требования к ПО сервера

Наличие установленного на сервер PHP и SQL.

Хостинг.

Домен.

4.2. Требования к оформлению

- Web-дизайн должен обеспечивать быстрый, удобный и интуитивно-понятный доступ к информации.
- Простая схема навигации.
- Кросс-браузерность.

4.3. Квалификация персонала, обслуживающего сайт

Общие навыки работы с Web-сайтами, добавление информации на сайт, редактирование информации.

4.4. Требования к системе администрирования

- Администратор имеет полномочия по изменению, удалению страниц, а также изменению самой БД и контента сайта.
- Администратор имеет право на удаление информации и модификации.

5. Структура сайта

1. Начальная страница

- Переход к просмотру ресурсов без авторизации.
- Информация об архиве.
- БД автомобильного ассортимента.

2. Описание и образцы автомобилей

- Страница для добавления новых автомобилей.
- Страница просмотра автомобилей.

3. О нас

Компания по бронированию автомобилей AutoRent предлагает широкий ассортимент автомобилей для каждого сегмента или уровня бюджета. У нас лучшие юристы, самое быстрое оформление заказа.

6. Языковые версии

- Русскоязычный сайт.

7. Группы пользователей

- Администратор.
- Менеджер — обладает правами администратора.
- Клиенты могут ознакомиться с характеристиками автомобилей или заказать обратный звонок, чтобы менеджер проконсультировал клиентов.

8. Дизайн сайта

На главной странице сайта (рис. 13) представлено меню видеосопровождения автомобилей.

Рис. 13. Главная страница сайта

На странице «оформление заказа» клиенты могут оформить заявку на бронирование автомобиля (рис. 14).

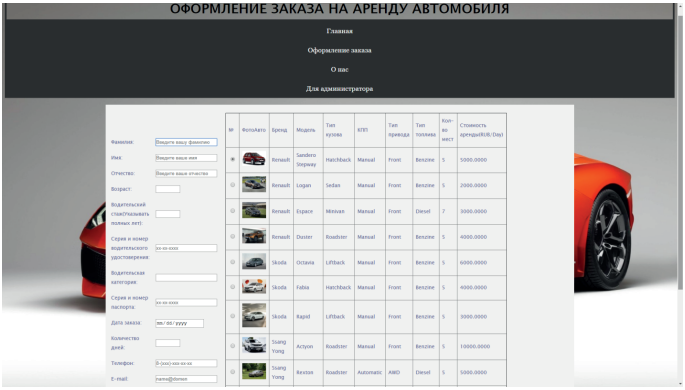


Рис. 14. Оформление заказа

Ссылка «О нас» позволяет ознакомиться с описанием компании аренды машины (рис. 15).

Рис. 15. Описание компании аренды машины

На рис. 16 и 17 представлены выбор комплектации автомобиля и местоположение компании.

Рис. 16. Выбор комплектации автомобиля

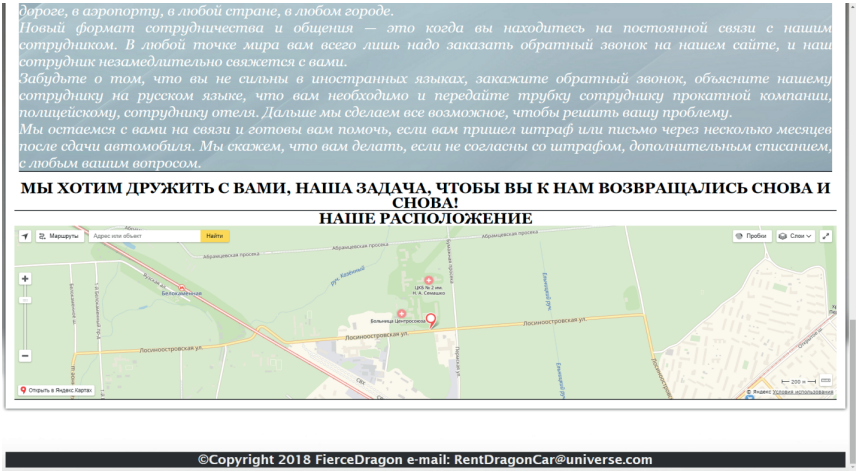


Рис. 17. Местоположение компании

9. Навигация сайта

На каждой странице должны быть ссылки на следующие страницы:

- главная страница;
- оформление заказа;
- о нас.

10. Описание страниц сайта

10.1. Статические страницы

- Начальная страница

Информация об ассортименте в БД. Описание и образцы архивных материалов.

Навигация по текстам осуществляется по следующим рубрикам:

- главная;
- оформить заказ;
- о нас.

10.2. Динамические страницы

- Страница добавления новых материалов.
- Страница добавления автомобилей.
- Страница добавления новой информации.
- Страница БД.

11. Функционал сайта

На сайте бронирования автомобиля можно сделать заказ бронирования автомобиля любого уровня бюджета. Доступ к просмотру интернет-проката предусмотрен для всех желающих.

12. Контент (наполнение) сайта

Аудио-, видео- и текстовые материалы, изображения. Поддерживаются наиболее известные форматы: mp3, wav, wma, doc, docx, txt, avi, wmv, jpeg, png.

Литература

1. Федеральный закон Российской Федерации от 27 июля 2006 г. № 149-ФЗ. «Об информации, информационных технологиях и о защите информации» (последняя редакция). — Текст: электронный // КонсультантПлюс: [сайт]. — URL: http://www.consultant.ru/document/cons_doc_LAW_61798/ (дата обращения 30.12.2019).
2. ГОСТ 24.202-80. Требования к содержанию документа «Технико-экономическое обоснование создания АСУ».
3. ГОСТ 34.201-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем.
4. ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированных систем.
5. ГОСТ 34.003-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения.
6. ГОСТ 34.601-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадия создания.

7. ГОСТ 34.603-92. Виды испытаний автоматизированных систем.
8. ГОСТ Р ИСОМЭК 9126-93. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению.
9. ГОСТ Р ИСО/МЭК 12207-99 Информационная технология. Процессы жизненного цикла программных средств.
10. ГОСТ РВ 51987-2002. Информационная технология. Комплекс стандартов на автоматизированные системы. Требования и показатели качества функционирования информационных систем (ИС). Общие положения.
11. ГОСТ Р ИСО 15288-2005 Информационная технология. Системная инженерия. Процессы жизненного цикла систем.
12. ГОСТ Р 57100-2016/ISO/IEC/IEEE 42010:2011. Системная и программная инженерия. Описание архитектуры.
13. *Вигерс Карл, Битти Джой*. Разработка требований к программному обеспечению. 3-е изд., доп. / пер. с англ. — М.: Русская редакция; СПб.: БХВ-Петербург, 2014. — 736 с.
14. Объектно-ориентированный анализ и проектирование с примером приложений /Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. — М.: Вильямс, 2017. —720 с.
15. *Буч Г., Якобсон А., Рамбо Дж.* UML. Классика CS. Издание второе. — СПб.: Питер, 2006. — 736 с.
16. *Галямина И.Г*. Управление процессами. — СПб.: Питер, 2013. — 304 с.
17. *Дейт К. Дж*. Введение в системы баз данных. 8-е издание / пер. с англ. — М.: Вильямс, 2005. — 1328 с.
18. *Ларман К.* Основы проектирования информационных систем Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку. — М.: Вильямс, 2013. — 736 с.
19. *Леоненков А.В*. Самоучитель UML 2. — СПб.: БХВ-Петербург, 2007. — 576 с.

20. *Маклаков С.В., Туманов В.Е.* Проектирование реляционных хранилищ данных. — М.: Диалог-МИФИ, 2007. — 336 с.
21. *Новиков Ф.А., Иванов Д.Ю.* Моделирование на UML. Теория, практика, видеокурс. — СПб.: Профессиональная литература, 2010. — 640 с.
22. *Рамбо Дж., Блаха М.* UML 2.0. Объектно-ориентированное моделирование и разработка. — СПб.: Питер, 2007. — 544 с.

Учебное издание

Татьяна Викторовна **Истомина**
Елена Валерьевна **Петрунина**
Александр Анатольевич **Белоглазов**
Эльмин Вагифович **Байрамов**

Разработка моделей сложных систем на языке UML

Учебно-методическое пособие

Ответственный редактор	С.А. Бобко
Редактор/корректор	Ю.Ф. Кравчинская
Технический редактор	К.А. Антонов
Компьютерная верстка	К.А. Антонов

Подписано в печать 25.12.2019. Формат 60x84 $^{1}/_{16}$.
Бумага офисная. Гарнитура *Times New Roman*. Печ. лист 6,5.
Тираж 500 экз. Заказ № 42.

Московский государственный гуманитарно-экономический университет
107150, Москва, ул. Лосиноостровская, д. 49.
Отпечатано в типографии МГГЭУ по технологии CtP.