Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Оренбургский государственный университет»

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Т.М. Зубкова

ФОРМИРОВАНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ ДЛЯ РАЗРАБОТКИ ПРОГРАММНЫХ ПРОДУКТОВ

Рекомендовано к изданию Редакционно-издательским советом федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Оренбургский государственный университет» в качестве методических указаний для студентов, обучающихся по программам высшего образования направления подготовки 09.03.04 Программная инженерия

Оренбург

2015

УДК 681.3 (07)
ББК 32.973.26-018я73
З 91

Рецензент – профессор, доктор технических наук В.И. Чепасов

Зубкова, Т.М.

Формирование технического задания для разработки программных
З 91 продуктов: методические указания /Т.М. Зубкова; Оренбургский гос. ун-т.- Оренбург: ОГУ, 2015. – 47 с.

Методические указания для выполнения практических заданий по дисциплине «Разработка и анализ требований» предназначены для оказания помощи студентам при выполнении индивидуальных заданий. Данная дисциплина входит в базовую часть профессионального цикла дисциплин бакалавров очной формы обучения по направлению 09.03.04 – «Программная инженерия» с профилем подготовки «Разработка программно-информационных систем».

В методических указаниях изложены задания и теоретические основы для их выполнения.

УДК 681.3 (07)
ББК 32.973.26-018я73

Содержание

Введение

Программная инженерия есть применение определенного систематического измеримого подхода при разработке, эксплуатации и поддержке программного обеспечения [IEEE Std 610.12-1990, IEEE – Institute of Electrical and Electronic Engineering – институт инженеров по электротехнике и радиоэлектронике Standard Glossary of Software Engineering Terminology].

Методические указания предназначены для выполнения практических заданий по курсу «Разработка и анализ требований».

Основными **задачами**, решаемыми в процессе освоения дисциплины, являются:

- обучить обнаруживать или выявлять требования, используя различные методы;
- обучить применять методы анализа, такие как анализ потребностей, анализ целей и вариантов использования;
- обучить оценивать требования в соответствии с критериями выполнимости, ясности, отсутствием неоднозначностей и т.д.;
- обучить представлять функциональные и нефункциональные требования для различных типов систем, используя формальные и неформальные методы;
- обучить определять и измерять атрибуты качества;
- обучить вести переговоры с различными заинтересованными лицами для достижения согласия по множеству требований.

Процесс изучения дисциплины направлен на формирование элементов следующих компетенций в соответствии с ФГОС ВПО и ООП ВПО по данному направлению подготовки 09.04.04 «Программная инженерия»:

а) общекультурных (ОК):

- владеть культурой мышления, способность к обобщению, анализу, восприятию информации, постановке цели и выбору путей ее достижения (ОК-1);
- готов к кооперации с коллегами, работе в коллективе (ОК-3);

- умеет использовать нормативно-правовые документы в своей деятельности (ОК-5);
- стремится к саморазвитию, повышению своей квалификации и мастерства (ОК-6).

б) профессиональных (ПК):

- понимание основных концепций, принципов, теорий и фактов, связанных с информатикой (ПК-1);
- способность к формализации в своей предметной области с учетом ограничений используемых методов исследования (ПК-2);
- готовность к использованию методов и инструментальных средств исследования объектов профессиональной деятельности (ПК-3);
- готовность обосновывать принимаемые проектные решения, осуществлять постановку и выполнение экспериментов по проверке их корректности и эффективности (ПК-4);
- умение готовить презентации, оформлять научно-технические отчеты по результатам выполненной работы, публиковать результаты исследований в виде статей и докладов на научно-технических конференциях (ПК-5);
- способность формализовать предметную область программного проекта и разработать спецификации для компонентов программного продукта (ПК-6).

Цель преподавания дисциплины «Разработка и анализ требований» по направлению подготовки 09.04.04 – «Программная инженерия» создание теоретической основы для проведения анализа и разработки требований к проектированию программного обеспечения различных типов.

В результате освоения дисциплины обучающийся должен:

Знать:

- основные факты, концепции, принципы и теории, связанные с информатикой;
- теоретические основы архитектуры и программной организации вычислительных и информационных систем;
- основы моделирования и анализа программных систем, разработки, выявления, спецификации и управления требованиями.

Уметь:

- разрабатывать и специфицировать требования;
- работать с современными системами программирования;
- оценивать бюджет, сроки и риски разработки программ.

Владеть:

- языками процедурного объектно-ориентированного программирования;
- навыками разработки и отладки программ на алгоритмических языках программирования;
- методами конструирования программного обеспечения и проектирования человеко-машинного интерфейса;
- методами и средствами разработки и оформления технической документацией.

Приобрести опыт деятельности в разработке требований к разработке и проектированию программного обеспечения.

1 Практическое занятие №1. Введение в разработку и анализ требований

На основании изучения предметной области, согласно варианту выполнить следующее:

1. Определить концепцию программного продукта.

Провести интервью с представителем заказчика (инвестором). Определить желания и потребности, определить инструменты разработки и поддержки, определить конфигурацию оборудования (задать не менее пяти вопросов). Создать черновик графического интерфейса пользователя.

2. Осуществить сбор требований с предполагаемым заказчиком.

Методические рекомендации для выполнения практической работы

Требования к программному обеспечению – совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащих реализации.

Этапы сбора и анализа требований

Процесс работы с требованиями к продукту можно разделить на 4 этапа:

- Определение концепции продукта.
- Сбор требований.
- Анализ требований.
- Проектирование системы

Определение концепции продукта

На этапе определения концепции продукта, проводится работа с его инвестором, целью которой является выработка единого видения будущего продукта (рис.1). По окончанию этого этапа производится вывод о том, будет ли этот продукт разрабатываться или нет.

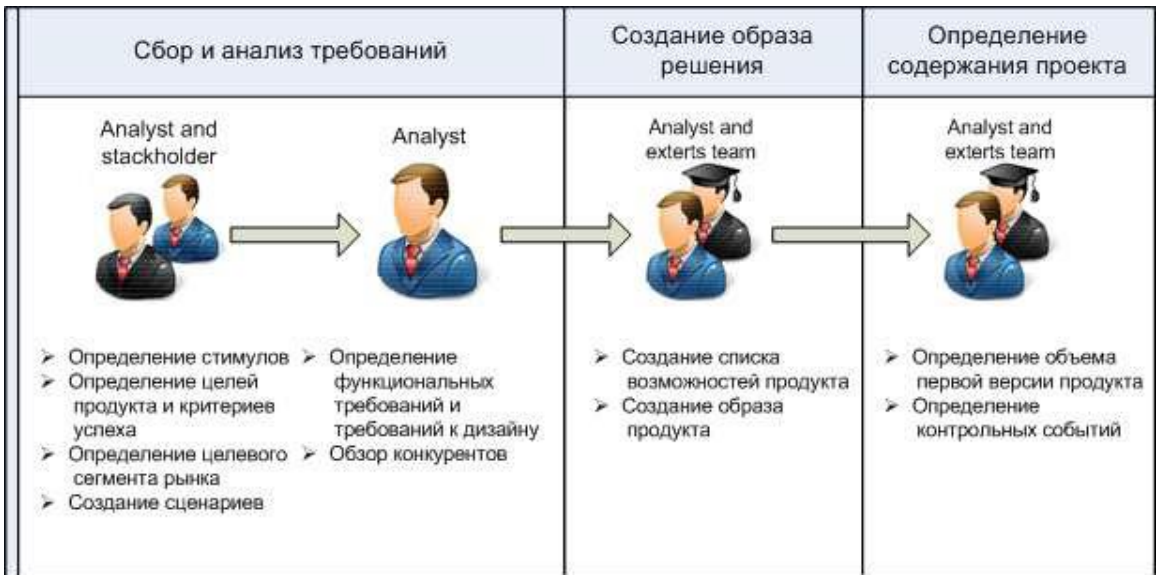

Рисунок 1 – Этапы разработки концепции продукта

Результатом разработки концепции является *Product Vision Document* (документ образа продукта, если продукт разрабатывается под заказ) или *Marketing Requirement Document* (документ рыночных требований, если продукт предназначен для открытого рынка). Эти документы содержат подробную информацию о требованиях заказчика, возможностях продукта, которые должны удовлетворять эти требования, а также ориентировочные сроки его реализации и бюджет. Различием между PVD и MRD является то, что в MRD обычно более детально описаны целевые сегменты рынка и сделан детальный обзор конкурентов.

По окончании разработки концепции продукта делается вывод о целесообразности разработки продукта.

Сбор требований

На этапе сбора требований основная работа ведется с заказчиком системы и её будущими пользователями. Цель этапа – точно определить функции продукта и способы его интеграции в существующие процессы.

Качественное выполнение работ на этом этапе гарантирует то, что будущий продукт будет соответствовать ожиданиям заказчика. Четкая расстановка приоритетов обеспечивает реализацию наиболее востребованной функциональ-

ности и исключение второстепенной/невостребованной функциональности, что сэкономит бюджет и сроки.

Первым и самым важным этапом в разработке продукта является *сбор бизнес требований*. Цель этой работы – определить основные требования бизнеса (исходные данные, истинные цели, которым должен служить продукт и проблемы, которые нужно преодолеть).

Для продуктов под заказ и продуктов для открытого рынка процесс сбора бизнес требований существенно различается.

Продукт под заказ – заказчик определен с самого начала, ему известны начальные предпосылки (стимулы) для инициации проекта и проблемы, которые продукт должен решать. В связи с этим, сбор требований начинается с определения исходных предпосылок, целей продукта и описания сценариев работы пользователей с будущим продуктом. Анализ конкурентных продуктов, которые могут удовлетворять схожие сценарии, делается в самом конце.

Продукт для открытого рынка – сектор рынка с самого начала может быть не определен, а цели продукта основываются на конкурентном анализе.

В результате, сразу за определением исходных предпосылок (стимулов) идет обзор конкурентов, далее идет определение целевого сегмента рынка и потребностей его заказчиков и только после этого определяются цели продукта и критерии его успеха. Последовательность задач, выполняемых на этапе сбора бизнес требований для продукта под заказ и для открытого рынка, приведена в таблице 1.

Анализ требований

На этапе анализа требований проходит структуризация уже собранных ранее требований. Цель этапа – предоставить четкий список не дублируемых требований к системе, которые должны быть выделены из избыточных и частично дублирующихся сценариев и пользовательских историй, которые были полученных на предыдущем этапе.

Таблица 1 – Последовательность задач, выполняемых на этапе сбора бизнес требований

Продукт под заказ	**Продукт для открытого рынка**
Определение исходных стимулов	Определение исходных стимулов
Определение целей продукта и критериев успеха	Обзор конкурентов
Определение потребностей клиента	Определение целевого сегмента рынка
Обзор конкурентов	Определение потребностей клиента
	Определение целей продукта и критериев успеха

Правильно сгруппированные требования помогут обойтись минимальным количеством функционала для удовлетворения максимально большего количества целей, а это, в свою очередь, поможет сэкономить бюджет и не даст расползтись рамкам проекта.

В течение некоторого времени проходили споры относительно того, кому «принадлежат» требования: заказчику или разработчикам. Для решения этого вопроса разделяют анализ требований на два уровня. Первый уровень документирует желания и потребности заказчика и пишется на языке, понятном заказчику. Результаты иногда называют требованиями заказника, *C-требованиями.* Первичной аудиторией для C-требований будет сообщество заказчиков, а уже вторичной – сообщество разработчиков. Второй уровень документирует требования в специальной, структурированной форме. Эти документы называются требованиями разработчика, или *D – требованиями* (детальные требования). Первичной аудиторией для D-требований будет сообщество разработчиков, а уже вторичной – сообщество заказчиков.

Проектирование системы

Целью всех предыдущих этапов был сбор информации о том, кому и зачем необходим будущий продукт. Этап проектирования – это первый этап, на котором группа разработки принимает проектные решения о том, какую функ-

циональность будет нести продукт, чтобы удовлетворить потребности пользователей.

Результатом этого этапа является законченное техническое задание к продукту. Оно должно содержать полное описание поведения будущего продукта и не содержать неоднозначностей и вопросов.

Практическое задание №2. Описание С-требований

1. Определить целевой сегмент рынка.
2. Определить бизнес роли пользователей и составить сценарии работы.

Каждый сценарий должен содержать:

- Имя конкретного заказчика или его профиль (в качестве заголовка).
- Информацию обо всех типах пользователей, которые будут работать с программным продуктом (ПП).
- Все процессы, которые будут затрагивать продукт.
- Операционная среда, в которой будет использоваться продукт.
- Требования к дизайну: операционная система, приложения, с которыми интегрируется будущий ПП, форматы ввода вывода.
- Приоритет. Зависит от того, насколько важен этот заказчик или как много покупателей будут попадать под профиль клиента, описанного в сценарии.

3. Определить функциональные требования.
4. Определить требования к дизайну (не функциональные требования).
5. Создать образ продукта.
6. Построить диаграмму потоков данных (использовать средства BPwin).

Методические рекомендации для выполнения практической работы

Первым и самым важным этапом в разработке продукта является *сбор бизнес требований*. Цель этой работы – определить основные требования бизнеса (исходные данные, истинные цели, которым должен служить продукт и проблемы, которые нужно преодолеть).

Для продуктов под заказ и продуктов для открытого рынка процесс сбора бизнес требований существенно различается.

Продукт под заказ – заказчик определен с самого начала, ему известны начальные предпосылки (стимулы) для инициации проекта и проблемы, которые продукт должен решать. В связи с этим, сбор требований начинается с определения исходных предпосылок, целей продукта и описания сценариев работы пользователей с будущим продуктом. Анализ конкурентных продуктов, которые могут удовлетворять схожие сценарии, делается в самом конце.

Продукт для открытого рынка – сектор рынка с самого начала может быть не определен, а цели продукта основываются на конкурентном анализе.

В результате, сразу за определением исходных предпосылок (стимулов) идет обзор конкурентов, далее идет определение целевого сегмента рынка и потребностей его заказчиков и только после этого определяются цели продукта и критерии его успеха.

Определение целевого сегмента рынка

Этот элемент является частью MRD и, как правило, требуется только для продуктов, ориентированных на широкий рынок. Принято сегментировать рынок следующим образом:

Рынок домашних пользователей (может быть также разделен на обычных и продвинутых пользователей).

Рынок корпоративных пользователей.

1. *SMB* (Small and Medium Business) – компании, насчитывающие от 1 до 250 сотрудников. Также может быть разделен на Micro (или Soho) – 1–10 сотрудников, Small – 10–25 и Medium – 25–250.

2. *Large* – компании, насчитывающие 250–2500 сотрудников.

3. *Corporation* – корпорации с числом сотрудников более 2500.

Это разделение принято использовать в связи с тем, что требования, налагаемые пользователями, которые принадлежат к разным сегментам рынка, кардинально отличаются. Это касается как функциональности, способов администрирования, так и вопросов лицензирования и поддержки продукта. Практически невозможно создать продукт, который бы подошел одновременно домаш-

ним пользователям и в тоже время мог бы эксплуатироваться в крупной компании.

Для того чтобы правильно выбрать сегмент рынка, необходимо определить требования, налагаемые каждым из сегментов рынка с учетом предметной области продукта.

Теперь, когда определен целевой сегмент для вашего продукта, следует определить кто, какие проблемы и в каких условиях будет решать при помощи будущего продукта.

Создание сценариев

Наиболее эффективным способом получения ответа на эти вопросы является определение сценариев работы пользователей с будущим продуктом. Сценарий – это совокупность всех процессов, в которых будет участвовать продукт, а также описание окружения, в котором его планируется использовать. Сценарий не должен являться описанием работы отдельного пользователя для достижения конкретной цели. Его ценность состоит в том, что он описывает способы взаимодействия с продуктом всех его пользователей одновременно на протяжении всего цикла эксплуатации продукта. Таким образом, сценарий гарантирует отсутствие взаимоисключающих требований к продукту и дает возможность легко убедиться, что никто и ничто не забыто. Для проверки сценария надо всего лишь проанализировать его выполнение всеми заинтересованными лицами (проиграть его).

Для продуктов под заказ сценарии использования продукта формируются самим заказчиком. Как правило – сколько заказчиков, столько и сценариев. Наиболее эффективным методом является живой диалог с заказчиком, в котором аналитик задает наводящие вопросы (пытается разговорить заказчика), а заказчик отвечает на них. Если личный контакт невозможен, как правило, помогают вопросники, содержащие «нужные» вопросы, по которым заказчику легче будет написать сценарий.

Для открытого рынка сначала определяются профили будущих клиентов продукта, а затем для каждого из них создается подробный сценарий его использования. Аналитик может описывать сценарии самостоятельно, используя информацию из личного опыта или открытых источников. Другой вариант, позволяющий достичь явного преимущества – найти клиентов или компании, подходящие под ранее определенные профили и получить сценарии непосредственно от них.

Важно не путать понятие клиента и пользователя. Клиентом может являться компания, в которой множество сотрудников будут являться пользователями системы. В таком случае, профиль клиента описывает характерные черты и проблемы компании, а профиль пользователя (описываемый позднее) – характеристики её сотрудников.

Благодаря использованию сценариев акцент делается на реальные потребности конкретных пользователей системы и лишь, потом определяется необходимый функционал продукта. Каждый сценарий содержит все процессы, в которых планируется использовать продукт.

Функциональные бизнес требования описывают потребность (заказчика/покупателя), которую должен удовлетворить будущий продукт. Основной вопрос, на который должны отвечать бизнес требования – зачем та или иная функциональность.

Требования к дизайну описывают операционную среду, в которой будет функционировать продукт, интерфейсы взаимодействия с пользователем или другими системами, форматы хранения и передачи данных, а также требования к надежности, производительности, обслуживанию и доступности системы. Эти требования в значительной степени влияют на средства разработки, архитектуру и используемые технологии при разработке продукта, а значит и на конечный бюджет и сроки продукта. Поэтому крайне важно как можно более точно определить важнейшие требования к дизайну именно на стадии определения концепции.

Разбивая требования на отдельные элементы, основным критерием должна быть возможность реализации этих требований отдельно друг от друга (реализовать первое и не реализовать второе). Если требования сильно зависят друг от друга, то должны быть реализованы вместе и их лучше объединить. Цель разделения требований на составные части – получение возможности принимать решения о целесообразности реализации каждого требования в отдельности и назначать им различные приоритеты, что в дальнейшем обеспечит гибкость в определении списка требований для первой и всех последующих итераций продукта или исключении из текущей версии продукта. Это критично для продуктов, бюджет и сроки которых строго определены и не могут быть пересмотрены.

Создание образа продукта

На протяжении всего жизненного цикла разработки «Образ продукта» должен являться дорожной картой, которой нужно пользоваться, чтобы не сбиться с пути. В силу этого он должен содержать самые главные данные и его описание не должно быть больше половины страницы. Для достижения этой цели лучше всего подходит следующий шаблон, который содержит все необходимые ключевые вопросы и тезисы:

- «для» – целевая аудитория покупателей;
- «который» – положение о потребностях или возможностях;
- «этот» – имя продукта;
- «является» – категория продукта;
- «который» – ключевое преимущество, основная причина для покупки или использования;
- «в отличие от» – основной конкурирующий продукт, текущая система или текущий бизнес-процесс;
- «наш продукт» – положение об основном отличии и преимуществе нового продукта.

Список возможностей должен полностью соответствовать образу решения. Если между образом решения и списком возможностей есть расхождения, вы должны это исправить (модернизировать список возможностей или образ решения).

Практическое задание №3. Определение основных профилей пользователей

1. На основании пользовательских профилей разработать черновой вариант пользовательского интерфейса.

2. Составить план работы после анализа С-требований.

Методические рекомендации для выполнения практической работы

Для того чтобы продукт был удобен пользователям и делал «то, что надо», сначала надо определить, кто же им будет пользоваться.

На практике, даже для домашних продуктов очень сложно определить «среднего пользователя», чтобы на основе его потребностей проектировать продукт. Для корпоративных продуктов это попросту невозможно: директор будет пользоваться одной функциональностью, его секретарь другой, а бухгалтер третьей. По этой причине перед началом сбора требований должны быть определены основные профили пользователей (рис.1).

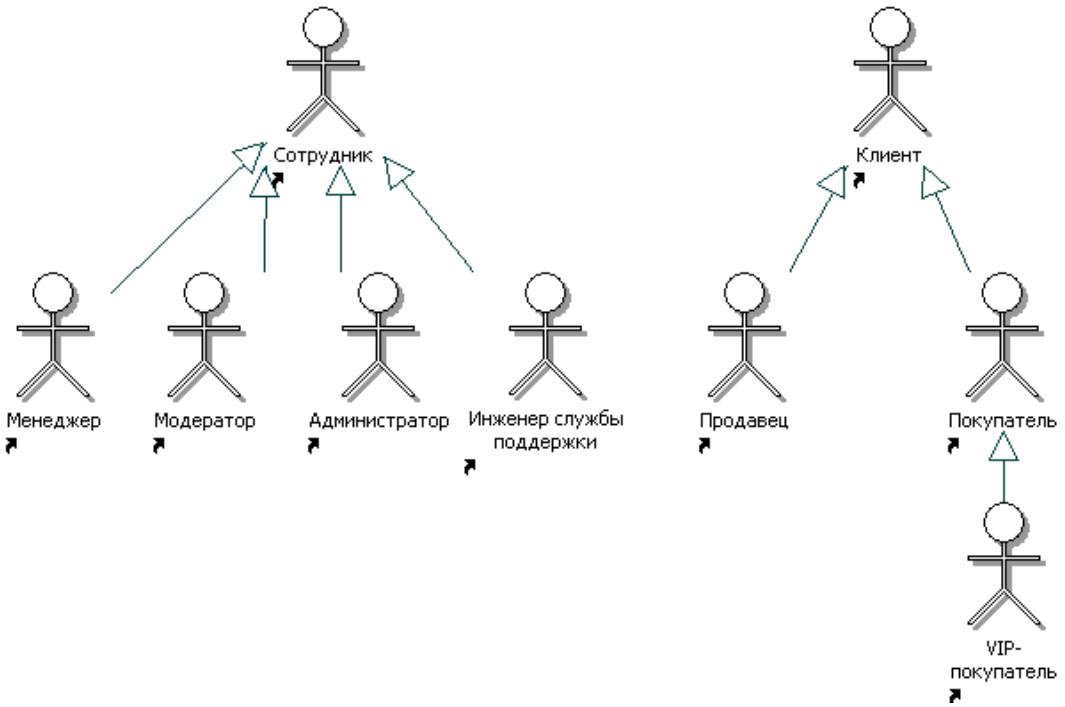

Рисунок 1 – Пример диаграммы профилей пользователей продукта

Профили пользователей явно или неявно уже должны содержаться в сценариях, поэтому все, что нужно сделать – это выделить их. Для домашних продуктов разделение обычно производится, исходя из уровня подготовленности пользователя и его интересов, для корпоративных продуктов основным критерием является специализация работника и круг его обязанностей.

Шаги разработки пользовательских интерфейсов

Предлагается 11 этапов разработки пользовательских интерфейсов.

1. Узнайте своего пользователя (C) (обработка С-требований).
2. Поймите назначение проектируемой системы (C).
3. Примените принципы хорошего экранного дизайна (C, D).
4. Подберите подходящий тип окон (C, D).
5. Разработайте системные меню (C, D).
6. Выберите соответствующие аппаратные устройства управления (C).
7. Выберите соответствующие экранные элементы управления (C).
8. Организуйте и создайте раскладку окон (C, D).
9. Выберите подходящие цвета (D).
10. Создайте осмысленные значки (C, D).
11. Предоставьте эффективные сообщения, обратную связь и руководство (D).

Шаг 1 (знакомство с пользователем). На этом шаге рекомендуется оценить общество конечных пользователей программы. В общих чертах основные факторы намечены в табл.1.

Шаг 2 (понимание назначения). На этом шаге от дизайнера требуется понимать цель конкретного предлагаемого пользовательского интерфейса в свете общего назначения программы. Например, если целью является инвентаризация склада, то пользовательский интерфейс может отражать план склада. Последовательность экранов может отражать способ, которым обычно пользователи выполняют свои задания вручную.

Таблица 1 – Критерии, по которым оцениваются потенциальные пользователи программы

Характеристика	Градации
Уровень знаний и опыт	
Компьютерная грамотность	Высокий
	Средний
	Низкий → объяснить каждый термин
Системный опыт	Высокий
	Средний
	Низкий → представить примеры и анимацию
Опыт работы с подобными программами	Высокий
	Средний
	Низкий → представить примеры и анимацию
Образование	Ученая степень
	Колледж
	Школа → использовать термины 12-го класса
Уровень чтения	>12 лет в школе/5-12/
	<5 → использовать очень простой язык
Машинопись	135 слов в минуту / 55/ [10 → представить небольшие поля для ввода текста, примеры, уделить особое внимание формам для заполнения]
Физические характеристики пользователя	
Возраст	Молодой/ среднего возраста/ пожилой
Пол	Мужской / женский
Развитие рук	/Левша/ Правша / владеющий одинаково обеими руками
Физические недостатки	Слепой / дефекты зрения / глухой / моторные недостатки

Продолжение таблицы 1

Характеристики заданий и работы пользователя	
Способ использования этой программы	По усмотрению/[обязательная → сделать программу интересной в использовании]
Частота использования	Постоянная / частая / случайная / [разовая → предоставить всю справочную информацию с каждым экраном]
Коэффициент текучести кадров	Низкий/ средний/[высокий → представить всю справочную информацию с каждым экраном]
Важность задания	Высокая / средняя / [низкая → сделать интересной в использовании]
Повторяемость задания	Низкая / средняя / [высокая → автоматизировать как можно больше шагов, предоставить разнообразие в представляемых данных, предоставить возможности обучения]
Предварительное обучение	Нет/самостоятельное изучение по справочникам / [интенсивное → предоставить интерактивную систему обучения]
Категория работы	Администратор/менеджер/ профессионал / секретарь / [клерк и т. д. → использовать язык, примеры и описания, знакомые обычному клерку]
Психологические характеристики пользователя	
Вероятное отношение к работе	Положительное / безразличное / отрицательное
Вероятные мотивации	Высокие /средние / [низкие → сделать приложение особенно привлекательным]
Стиль процесса познания	Словесный или [пространственный → подчеркнуть геометрический вид] Аналитический или [интуитивный → подчеркнуть символы в тексте] Конкретный или [абстрактный → разработать обобщения]

Шаг 3 (понимание принципов хорошего экранного дизайна). Перечислим некоторые основные элементы хорошего экранного дизайна.

1. Убедитесь в единообразии экранов приложения, а также в логичности каждого отдельно.

- Соглашения; процедуры; местоположение.

2. Сделайте предположение о том, откуда, обычно пользователь будет начинать работу.

- Часто «первый» элемент размещают в верхнем левом углу.
- 3. Сделайте навигацию как можно более простой:
- выровняйте похожие элементы;
- сгруппируйте похожие элементы;
- учтите границы вокруг похожих элементов.

4. Примените иерархию для подчеркивания порядка важности.

5. Примените принципы приятных визуальных эффектов:

- баланс, симметрия, регулярность, предсказуемость;
- простота, единообразие, пропорциональность, экономия.

6. Предоставьте подписи.

Шаг 4 (выбор подходящего типа окна). Цели каждого пользовательского интерфейса могут обслуживаться наиболее эффективно одним или двумя конкретными типами окон.

Шаг 5 (разработка системного меню). Ниже перечислены некоторые правила для создания главных меню.

- Сделать главное меню.
- Показать все уместные альтернативы (но только их).
- Привести структуру меню в соответствие со структурой задачи приложения.
- Минимизировать число уровней меню.

Шаг 6 (выбор подходящих устройств управления). Под устройствами управления здесь понимаются физические устройства, с помощью которых пользователи сообщают свои пожелания приложению. Сюда относятся джойстики, трекболы, графические планшеты, сенсорные экраны, мыши, микрофоны и клавиатуры.

Шаг 7 (выбор подходящих экранных элементов управления). Экранные элементы управления – это символы, появляющиеся на мониторе, с помощью которых пользователь передает программе вводимые данные и свои намерения.

Сюда относятся значки, кнопки, текстовые окна, списки и др. Правила организации экранных элементов управления в окне практически те же, что и для: дизайна экрана в общем. Их число также обычно варьируется: от пяти до девяти. Это число, однако, может быть увеличено в случае использования иерархии.

Шаг 8 (организация и планирование окон). Правила для компоновки многочисленных окон аналогичны правилам для дизайна одиночных окон (в том числе симметрия, пропорциональность и т.д.), но сюда включена также и организация окон – соприкасающиеся или каскадные.

Шаг 9 (выбор подходящих цветов). При использовании с умением и со вкусом цвет может обогатить экран. Использование цвета не делает автоматически пользовательский интерфейс более полезным или привлекательным, однако может легко испортить его. По выражению знаменитого дизайнера Поля Рэнда «цвет – это воплощение сложности». Инженеры-программисты, не сотрудничающие с профессиональными дизайнерами, должны быть очень умеренными и консервативными в использовании цветов. Сначала попробуйте черно-белую схему. Если есть очевидная потребность в этом, добавьте один цвет. Убедитесь, что это помогает пользователю. Серьезно подумайте перед тем, как добавлять больше цветов.

Результирующее расписание по работе над проектом может быть таким как показано в таблице 2.

Таблица 2 –Типичный план после анализа С-требований

	7 мая	1 мая	13июн	7июн	1июл	25июл	1 авг	5 авг	8 сен
Вехи	Законченная версия 0.1Х								
Разработка версии 0.1									
С-требования									
D-требования									
Архитектура									
Детальное проектирование									
Реализация									
Тестирование									
Разработка версии 0.2									
С-требования									
D-требования									
Архитектура									
Детальное проектирование									
Реализация									
Тестирование компонентов									
Интеграция									
Системное тестирование									

Практическое задание №4. Разработка детальных требований (D-требования)

1. На основании данных полученных от заказчика и сборе пользовательских историй провести:

- структурирование пользовательских историй;
- спецификацию требований.

2. Провести контроль функциональных требований и нефункциональных требований

3. На диаграмме последовательности показать реализации вариантов использования.

4. Разработать пользовательский интерфейс, на котором показать реализацию функциональных требований и эргономичность интерфейса.

Методические рекомендации для выполнения практической работы

Разработчикам программного обеспечения нужна база для проектирования и разработки. Эта база состоит из детальных требований. Их также называют конкретными требованиями, функциональными спецификациями, требованиями разработчика или D-требованиями. D-требования состоят из полного списка конкретных свойств и функциональности, которую должна иметь программа, сформулированных в подробностях. Каждое из этих требований пронумеровано, помечено и отслеживается по ходу разработки. D-требования должны быть согласованы с C-требованиями. Предполагается, что D-требования будут читать преимущественно разработчики.

Основной целью анализа требований является их систематизация и избавление от дублируемых данных. Это достигается за счет разделения пользовательских историй на отдельные пакеты по функциональному признаку и их иерархической структуризации.

По окончании этапа анализа требований, многостраничный документ, содержащий сотни пользовательских историй, будет разбит на части. Каждая часть будет освещать только необходимую функциональность, а в её основе будет стоять диаграмма вариантов использования, на основе которой можно будет легко увидеть все требуемые функции системы. Описание вариантов использования не будут дублироваться, а лишь дополнять друг друга.

Выделение пользовательских историй в отдельные пакеты

Первым этапом анализа требования является выделение пользовательских историй в отдельные пакеты требований. Для специализированных систем управления требованиями вроде Borland Caliber RM или Rational Request Pro, в которых все требования хранятся в базе данных, и представлены древовидным списком, пакетами являются элементы первого уровня, которые будут играть роль контейнера для всех остальных требований. При использовании текстовых документов пакетами будут являться отдельные файлы, возможно с общим индексом.

Главная цель формирования пакетов – упростить доступ к нужным данным, за счет того, что все варианты использования относящихся к определенной функциональности можно будет увидеть на одной странице (оглавление или диаграмма вариантов использования). В случае использования текстовых документов, пакеты также существенно упростят автору процесс последующего редактирования – не нужно будет блокировать весь документ на время редактирования, а пользователям документа будет легче узнать об изменениях (как правило, для этого используется секция «История изменений» в начале каждого документа). Для того чтобы достичь поставленной цели требуется добиться того, чтобы в один пакет входило 20–30 пользовательских историй.

Пакеты формируются из пользовательских историй, которые описывают схожую деятельность или способ достижения схожего результата. Как правило, всего они подчинены одному бизнес требованию.

Два подхода: Варианты использования или спецификации требований

Существует два основных метода проектирования – проектирование на основе вариантов использования и проектирование на основе требований.

Проектирование на базе вариантов использования считается более эффективным, так как этот метод позволяет не терять связь с пользовательскими историями и прекрасно иллюстрирует требуемое поведение системы в целом, а, следовательно, гарантирует востребованность всего функционала, который будет создан (очень расточительно и болезненно для разработчиков писать код, которым никогда не удастся воспользоваться). Но все, же есть условия, при которых аналитик может пренебречь этим методом в пользу проектирования на базе требований:

- Проектирование на основе требований следует предпочесть, если необходимо сократить затраты на стадию анализа до минимума, а количество пользовательских историй в пакете не велико (их содержимое можно просмотреть не дольше чем за 10 минут).
- Команда разработки не умеет или не хочет работать с вариантами использования и требует предоставление требований к системе в классическом виде.

Далее описаны работы, которые нужно произвести в рамках выбранного вами метода.

А. Варианты использования. Структурирование пользовательских историй

Как правило, у всех пользовательских историй в пакете есть одна или несколько главных целей, и есть история, которая описывает наиболее простой способ их достижения. Такая история называется – базовой, а описание её действия – базовый путь. Остальные истории описывают альтернативный способ достижения результата или содержат дополнительные действия для достижения специфического результата и по большому счету являются дополнениями к базовой истории.

Пользовательские истории это разновидность стандартных вариантов использования, с той лишь особенностью, что они описывают взаимодействие не с реальной, а с гипотетической системой. Это свойство пользовательских историй будет использовано в этой главе и все манипуляции в процессе структурирования будут производиться в соответствии с правилами и методами, разработанными для стандартных вариантов использования.

Б. Спецификация требований

Для нормальной разработки нужен список требований, который однозначно идентифицирует потребности пользователей и не имеет множественных повторений, которые присутствуют с избытком в пользовательских историях (если не произвести структурирование историй, описанное в предыдущем подходе). Кроме этого, для более гибкого проектирования необходим как можно более детализированный (раздробленный) список.

Извлечение требований из пользовательских историй

На этом этапе нужно выделить все уникальные результаты пользовательских историй в отдельные требования. Если результат истории оказался уникальным, то вы должны преобразовать его в требование (просто скопировать его тело, используя повелительное наклонение). Если же похожий результат уже был вынесен в требование в рамках другой пользовательской истории, то надо проанализировать их возможные отличия и, если они есть – модернизировать существующее требование или создать дополнительное дочернее требование/ограничение. Приоритет требования должен быть унаследован от родительской истории с максимальным приоритетом.

Кроме результата пользовательской истории следует проанализировать ее поток выполнения. Он может содержать уточнения/пожелания к поведению продукта, которые не плохо было бы учесть. Пожеланиям следует назначать приоритет ниже, чем приоритет родительской истории.

В результате этой работы должен быть получен древовидный список требований.

Типы D-требований

Существуют несколько типов требований.

1. Функциональные требования:

- функциональность приложения.

2. Нефункциональные требования.

1) Производительность:

- скорость;
- пропускная способность (трафик);
- использование памяти (оперативная память, жесткий диск).

2) Надежность и доступность.

3) Обработка ошибок.

4) Интерфейсные требования.

Как программа взаимодействует с пользователем и с другими программами.

5) Ограничения:

- точность;
- ограничения на инструменты и язык, например «должен использоваться C#»;
- ограничения проектирования;
- стандарты, которые должны быть использованы;
- платформы, которые должны быть использованы.

3. Обратные требования.

Чего программа не делает.

Диаграммы последовательности

Диаграммы последовательности являются графическим представлением передачи управления и особенно полезны для визуализации реализации вариантов использования.

Диаграммы последовательности заставляют нас рассуждать в терминах объектов. В такой диаграмме жизненный цикл каждого участвующего объекта показан вертикальной линией с именем объекта и указанием его класса вверху. Каждое взаимодействие между объектами отображается горизонтальной стрелкой от объекта, инициирующего взаимодействие, к объекту, выполняющему дальнейшие функции.

Способы организации детальных требований

D-требования можно организовать с помощью нескольких схем:

• по основным свойствам (предоставляемый вовне сервис, обычно определяется с помощью пар стимул-реакция). Этот способ организации часто воспринимают как «требования», имея в виду, что требования сгруппированы по различным свойствам программы. Это не предоставляет никакой систематической организации, поскольку позволяет переходить от свойства одной части программы к свойству абсолютно другой части программы;

• по режиму (например, системы управления радарами могут иметь тренировочный, нормальный и аварийный режимы);

• по вариантам использования (иногда еще называется по сценариям). Идея заключается в том, что большинство детальных требований являются частью варианта использования;

• по классу. Это объектно-ориентированный стиль, в этом способе организации мы классифицируем требования по классам;

• по иерархии функций (то есть путем разбиения программы на множество высокоуровневых функций и последующего разбиения их на подфункции и т. д.) Например, требования для программы домашнего бюджета можно разбить на (1) функции проверки, (2) функции сбережений и (3) функции инвести-

рования. Функции проверки могут затем быть разложены на функции чековой книжки, баланса счета, составление отчетов и т. д. Это традиционный способ упорядочивания детальных требований;

- по состояниям (то есть путем указания детальных требований, применимых к каждому состоянию). В частности, требования для программы, управляющей химическим процессом, лучше всего классифицировать по состояниям, в которых может находиться процесс (начало, реакция, охлаждение и т. д.). Внутри классификации каждого состояния перечислены события, влияющие на программу, находящуюся в конкретном состоянии.

Классификация по состояниям может быть уместна, если требования для каждого состояния сильно отличаются. Например, бухгалтерская система может вести себя по-разному в зависимости от ее состояний, таких как «Конфигурация», «Исполнение» или «Сохранение».

Заключение

В методических указаниях приведено поэтапное формирование требований к разработке программного обеспечения. На основании определения концепции программного средства и выявления пожеланий заказчика определяются сценарии работы и бизнес роли пользователей. Определяются функциональные и не функциональные требования к ПП, а также их контроль. На основании профилей пользователей разрабатывается, протопит пользовательского интерфейса, на котором показывается реализация функциональных требований и эргономичность с учетом принципов хорошего экранного дизайна: подходящего типа окон, системного меню и др.

В результате выполнения практических заданий студенты закрепляют теоретические знания и формируют практические навыки по разработке технического задания к программному продукту.

Представленный теоретический и справочный материал позволяет использовать методические рекомендации для разработки требований к ПП.

Список использованных источников

1 Кулямин, В. В. Технологии программирования. Компонентный подход: / В. В. Кулямин . - М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2011. - 463 с. : ил. ISBN 978-5-94774-544-3 (БИНОМ.ЛЗ), ISBN 5-9556-0067-1 (ИНТРУИТ.РУ)

2 Орлов, С. А. Технологии разработки программного обеспечения: разработка сложных программных систем: учеб. для вузов / С. А. Орлов .- 3-е изд. - СПб. [и др.] : Питер, 2004. - 527 с.: ил. ISBN 5-94723-145-X

3 Пилон, Д. Управление разработкой ПО/ Д. Пилон, Р. Майлз. – СПб.: Питер, 2011. – 464 с.: ил. ISBN 978-5-459-00522-6

4 Брауде, Э. Технология разработки программного обеспечения/ Э. Брауде. – СПб.: Питер, 2004. – 655 с.: ил. ISBN 5-94723-663-X

5 Гагарина, Л. Г. Технология разработки программного обеспечения: учеб. пособие для вузов по направлению "Информатика и вычислительная техника" / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул; под ред. Л. Г. Гагариной. - М. : ИД «ФОРУМ»: ИНФРА-М, 2008. – 400 с: ил. ISBN 978-5-8199-0342-1 (ИД «ФОРУМ») ISBN 978-5-16-003193-4 (ИНФРА-М)

6 Иванова, Г. С. Технология программирования: учебник для вузов / Г. С. Иванова .- 3-е изд., перераб. и доп. - М. : МГТУ им. Н.Э. Баумана, 2006. - 336 с.: ил. ISBN ISBN 5-7038-2077-4

Приложение А

Унифицированный язык моделирования

UML – стандартный язык для написания моделей анализа, проектирования и реализации объектно-ориентированных программных систем (Unified Modeling Language). UML может использоваться для визуализации, спецификации, конструирования и документирования результатов программных проектов. UML – это не визуальный язык программирования, но его модели прямо транслируются в текст на языках программирования (Java, C++, Visual Basic, Ada 95, Object Pascal) и даже в таблицы для реляционной БД.

Словарь UML образуют три разновидности строительных блоков: предметы, отношения, диаграммы.

Предметы – это абстракции, которые являются основными элементами в модели, отношения связывают эти предметы, диаграммы группируют коллекции предметов.

Предметы в UML

В UML имеются четыре разновидности предметов:

- структурные предметы;
- предметы поведения;
- группирующие предметы;
- поясняющие предметы.

Эти предметы являются базовыми объектно-ориентированными строительными блоками. Они используются для написания моделей.

Структурные предметы являются существительными в UML-моделях. Они представляют статические части модели – понятийные или физические элементы. Перечислим восемь разновидностей структурных предметов.

1. *Класс* – описание множества объектов, которые разделяют одинаковые свойства, операции, отношения и семантику (смысл). Класс реализует один или

несколько интерфейсов. Как показано на рисунке 1, графически класс отображается в виде прямоугольника, обычно включающего секции с именем, свойствами (атрибутами) и операциями.

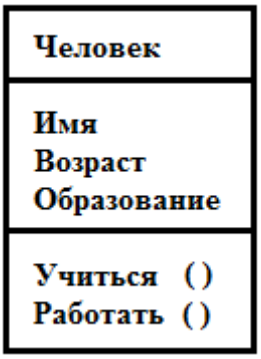

Рисунок 1 – Классы

2. *Интерфейс* – набор операций, которые определяют услуги класса или компонента. Интерфейс описывает поведение элемента, видимое извне. Интерфейс может представлять полные услуги класса или компонента или часть таких услуг. Интерфейс определяет набор спецификаций операций (их сигнатуры), а не набор реализаций операций. Графически интерфейс изображается в виде кружка с именем, как показано на рисунке 2. Имя интерфейса обычно начинается с буквы «I». Интерфейс редко показывают самостоятельно. Обычно его присоединяют к классу или компоненту, который реализует интерфейс.

Рисунок 2 – Интерфейсы

3. *Кооперация* (сотрудничество) определяет взаимодействие и является совокупностью ролей и других элементов, которые работают вместе для обеспечения коллективного поведения более сложного, чем простая сумма всех элементов. Таким образом, кооперации имеют как структурное, так и поведенческое измерения. Конкретный класс может участвовать в нескольких коопера-

циях. Эти кооперации представляют реализацию паттернов (образцов), которые формируют систему. Как показано на рисунке 3, графически кооперация изображается как пунктирный эллипс, в который вписывается ее имя.

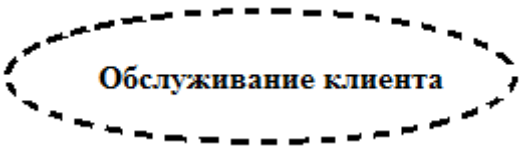

Рисунок 3 – Кооперации

4. *Актер* – набор согласованных ролей, которые могут играть пользователи при взаимодействии с системой (ее элементами Use Case). Каждая роль требует от системы определенного поведения. Как показано на рисунке 4, актер изображается как проволочный человечек с именем.

Рисунок 4 – Актеры

5. *Элемент Use Case* (Прецедент) – описание последовательности действий (или нескольких последовательностей), выполняемых системой в интересах отдельного актера и производящих видимый для актера результат. В модели элемент Use Case применяется для структурирования предметов поведения. Элемент Use Case реализуется кооперацией. Как показано на рисунке 5, элемент Use Case изображается как эллипс, в который вписывается его имя.

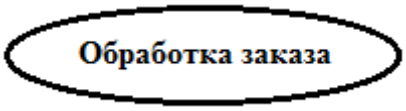

Рисунок 5 – Элементы Use Case

6. *Активный класс* – класс, чьи объекты имеют один или несколько про-

цессов (или потоков) и поэтому могут инициировать управляющую деятельность. Активный класс похож на обычный класс за исключением того, что его объекты действуют одновременно с объектами других классов. Как показано на рисунке 6, активный класс изображается как утолщенный прямоугольник, обычно включающий имя, свойства (атрибуты) и операции.

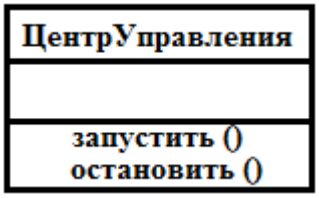

Рисунок 6 – Активные классы

7. *Компонент* – физическая и заменяемая часть системы, которая соответствует набору интерфейсов и обеспечивает реализацию этого набора интерфейсов. В систему включаются как компоненты, являющиеся результатами процесса разработки (файлы исходного кода), так и различные разновидности используемых компонентов (COM + компоненты, Java Beans). Обычно компонент – это физическая упаковка различных логических элементов (классов, интерфейсов и сотрудничеств). Как показано на рисунке 7, компонент изображается как прямоугольник с вкладками, обычно включающий имя.

Рисунок 7 – Компоненты

8. *Узел* – физический элемент, который существует в период работы системы и представляет ресурс, обычно имеющий память и возможности обработки. В узле размещается набор компонентов, который может перемещаться от узла к узлу. Как показано на рисунке 8, узел изображается как куб с именем.

Рисунок 8 – Узлы

Предметы поведения – динамические части UML-моделей. Они являются глаголами моделей, представлением поведения во времени и пространстве. Существует две основные разновидности предметов поведения.

1. *Взаимодействие* – поведение, заключающее в себе набор сообщений, которыми обменивается набор объектов в конкретном контексте для достижения определенной цели. Взаимодействие может определять динамику как совокупности объектов, так и отдельной операции. Элементами взаимодействия являются сообщения, последовательность действий (поведение, вызываемое сообщением) и связи (соединения между объектами). Как показано на рисунке 9, сообщение изображается в виде направленной линии с именем ее операции.

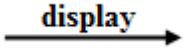

Рисунок 9 – Сообщения

2. *Конечный автомат* – поведение, которое определяет последовательность состояний объекта или взаимодействия, выполняемые в ходе его существования в ответ на события (и с учетом обязанностей по этим событиям). С помощью конечного автомата может определяться поведение индивидуального класса или кооперации классов. Элементами конечного автомата являются состояния, переходы (от состояния к состоянию), события (предметы, вызывающие переходы) и действия (реакции на переход). Как показано на рисунке 10, состояние изображается как закругленный прямоугольник, обычно включающий его имя и его подсостояния (если они есть).

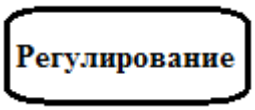

Рисунок 10 – Состояния

Эти два элемента – взаимодействия и конечные автоматы – являются базисными предметами поведения, которые могут включаться в UML-модели. Семантически эти элементы ассоциируются с различными структурными элементами (прежде всего с классами, сотрудничествами и объектами).

Группирующие предметы – организационные части UML-моделей. Это ящики, по которым может быть разложена модель. Предусмотрена одна разновидность группирующего предмета – пакет.

Пакет – общий механизм для распределения элементов по группам. В пакет могут помещаться структурные предметы, предметы поведения и даже другие группировки предметов. В отличие от компонента (который существует в период выполнения), пакет – чисто концептуальное понятие. Это означает, что пакет существует только в период разработки. Как показано на рисунке 11, пакет изображается как папка с закладкой, на которой обозначено его имя и, иногда, его содержание.

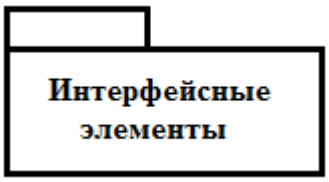

Рисунок 11 – Пакеты

Поясняющие предметы – разъясняющие части UML-моделей. Они являются замечаниями, которые можно применить для описания, объяснения и комментирования любого элемента модели. Предусмотрена одна разновидность поясняющего предмета – примечание.

Примечание – символ для отображения ограничений и замечаний, присоединяемых к элементу или совокупности элементов. Как показано на рисунке

12, примечание изображается в виде прямоугольника с загнутым углом, в который вписывается текстовый или графический комментарий.

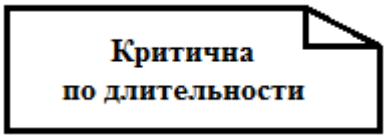

Рисунок 12 – Примечания

Отношения в UML

В UML имеются четыре разновидности отношений:

1) зависимость;

2) ассоциация;

3) обобщение;

4) реализация.

Эти отношения являются базовыми строительными блоками отношений. Они используются при написании моделей.

1. *Зависимость* – семантическое отношение между двумя предметами, в котором изменение в одном предмете (независимом предмете) может влиять на семантику другого предмета (зависимого предмета). Как показано на рисунке 13, зависимость изображается в виде пунктирной линии, возможно направленной на независимый предмет и иногда имеющей метку.

Рисунок 13 – Зависимости

2. *Ассоциация* – структурное отношение, которое описывает набор связей, являющихся соединением между объектами. Агрегация – это специальная разновидность ассоциации, представляющая структурное отношение между целым и его частями. Как показано на рисунке 14, ассоциация изображается в виде сплошной линии, возможно направленной, иногда имеющей метку и часто включающей другие «украшения», такие как мощность и имена ролей.

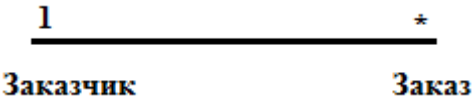

Рисунок 14 – Ассоциации

3. *Обобщение* – отношение специализации/обобщения, в котором объекты специализированного элемента (потомка, ребенка) могут заменять объекты обобщенного элемента (предка, родителя). Иначе говоря, потомок разделяет структуру и поведение родителя. Как показано на рисунке 15, обобщение изображается в виде сплошной стрелки с полым наконечником, указывающим

на родителя.

Рисунок 15 – Обобщения

4. *Реализация* – семантическое отношение между классификаторами, где один классификатор определяет контракт, который другой классификатор обязуется выполнять (к классификаторам относят классы, интерфейсы, компоненты, элементы Use Case, кооперации). Отношения реализации применяют в двух случаях: между интерфейсами и классами (или компонентами), реализующими их; между элементами Use Case и кооперациями, которые реализуют их. Как показано на рисунке 16, реализация изображается как нечто среднее между обобщением и зависимостью.

Рисунок 16 – Реализации

Диаграммы в UML

Диаграмма – графическое представление множества элементов, наиболее часто изображается как связный граф из вершин (предметов) и дуг (отношений). Диаграммы рисуются для визуализации системы с разных точек зрения, затем они отображаются в систему. Обычно диаграмма дает неполное представление элементов, которые составляют систему. Хотя один и тот же элемент может появляться во всех диаграммах, на практике он появляется только в некоторых диаграммах. Теоретически диаграмма может содержать любую комбинацию предметов и отношений, на практике ограничиваются малым количеством комбинаций, которые соответствуют пяти представлениям архитектуры ПС. По этой причине UML включает девять видов диаграмм:

1) диаграммы классов;

2) диаграммы объектов;

3) диаграммы Use Case (диаграммы прецедентов);

4) диаграммы последовательности;

5) диаграммы сотрудничества (кооперации);

6) диаграммы схем состояний;

7) диаграммы деятельности;

8) компонентные диаграммы;

9) диаграммы размещения (развертывания).

Диаграмма классов показывает набор классов, интерфейсов, сотрудничеств и их отношений. При моделировании объектно-ориентированных систем диаграммы классов используются наиболее часто. Диаграммы классов обеспечивают статическое проектное представление системы. Диаграммы классов, включающие активные классы, обеспечивают статическое представление процессов системы.

Диаграмма объектов показывает набор объектов и их отношения. Диаграмма объектов представляет статический «моментальный снимок» с экземпляров предметов, которые находятся в диаграммах классов. Как и диаграммы классов, эти диаграммы обеспечивают статическое проектное представление или статическое представление процессов системы (но с точки зрения реальных или фототипичных случаев).

Диаграмма Use Case (диаграмма прецедентов) показывает набор элементов Use Case, актеров и их отношений. С помощью диаграмм Use Case для системы создается статическое представление Use Case. Эти диаграммы особенно важны при организации и моделировании поведения системы, задании требований заказчика к системе.

Диаграммы последовательности и диаграммы сотрудничества – это разновидности диаграмм взаимодействия.

Диаграмма взаимодействия показывает взаимодействие, включающее набор объектов и их отношений, а также пересылаемые между объектами сообщения. Диаграммы взаимодействия обеспечивают динамическое представление системы.

Диаграмма последовательности – это диаграмма взаимодействия, которая выделяет упорядочение сообщений по времени.

Диаграмма сотрудничества (диаграмма кооперации) – это диаграмма взаимодействия, которая выделяет структурную организацию объектов, посылающих и принимающих сообщения. Диаграммы последовательности и диаграммы сотрудничества изоморфны, что означает, что одну диаграмму можно трансформировать в другую диаграмму.

Диаграмма схем состояний показывает конечный автомат, представляет состояния, переходы, события и действия. Диаграммы схем состояний обеспечивают динамическое представление системы. Они особенно важны при моделировании поведения интерфейса, класса или сотрудничества. Эти диаграммы выделяют такое поведение объекта, которое управляется событиями, что особенно полезно при моделировании реактивных систем.

Диаграмма деятельности – специальная разновидность диаграммы схем состояний, которая показывает поток от действия к действию внутри системы. Диаграммы деятельности обеспечивают динамическое представление системы. Они особенно важны при моделировании функциональности системы и выделяют поток управления между объектами.

Компонентная диаграмма показывает организацию набора компонентов и зависимости между компонентами. Компонентные диаграммы обеспечивают статическое представление реализации системы. Они связаны с диаграммами классов в том смысле, что в компонент обычно отображается один или несколько классов, интерфейсов или кооперацій.

Диаграмма размещения (диаграмма развертывания) показывает конфигурацию обрабатывающих узлов периода выполнения, а также компоненты, живущие в них. Диаграммы размещения обеспечивают статическое представление размещения системы. Они связаны с компонентными диаграммами в том смысле, что узел обычно включает один или несколько компонентов.

Механизмы расширения в UML

UML – развитый язык, имеющий большие возможности, но даже он не может отразить все нюансы, которые могут возникнуть при создании различных моделей. Поэтому UML создавался как открытый язык, допускающий контролируемые рас-. ширения. Механизмами расширения в UML являются:

- ограничения;
- теговые величины;
- стереотипы.

Ограничение (constraint) расширяет семантику строительного UML-блока, позволяя добавить новые правила или модифицировать существующие. Ограничение показывают как текстовую строку, заключенную в фигурные скобки {}. Например, на рисунке 17 введено простое ограничение на свойство *сумма* класса *Сессия Банкомата* – его значение должно быть кратно 20. Кроме того, здесь показано ограничение на два элемента (две ассоциации), оно располагается возле пунктирной линии, соединяющей элементы, и имеет следующий смысл – владельцем конкретного счета не может быть и организация, и персона.

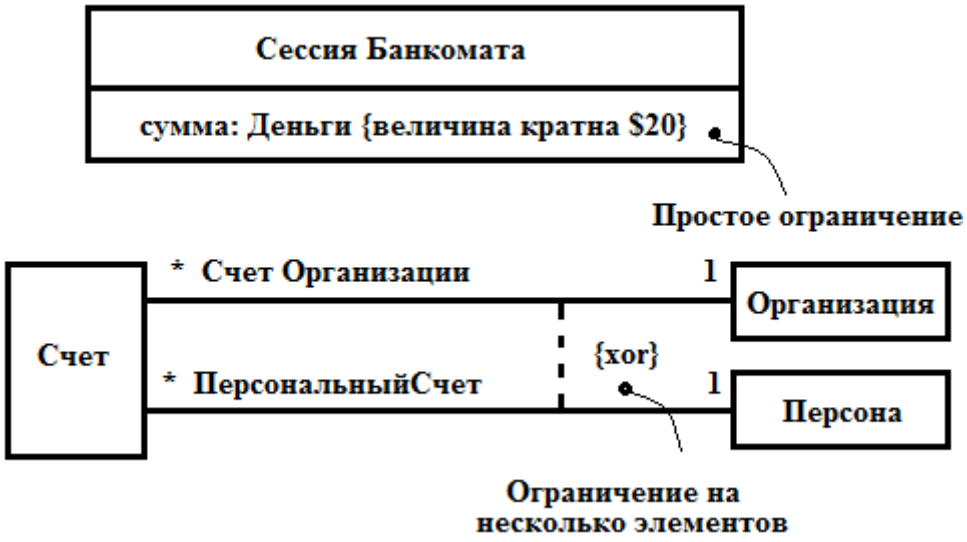

Рисунок 17 – Ограничения

Теговая величина (tagged value) расширяет характеристики строительного UML-блока, позволяя создать новую информацию в спецификации конкретного

элемента. Теговую величину показывают как строку в фигурных скобках {}. Строка имеет вид

имя теговой величины = значение.

Иногда (в случае предопределенных тегов) указывается только имя теговой величины.

Отметим, что при работе с продуктом, имеющим много реализаций, полезно отслеживать версию и автора определенных блоков. Версия и автор не принадлежат к основным понятиям UML. Они могут быть добавлены к любому строительному блоку (например, к классу) введением в блок новых теговых величин. Например, на рисунке 18 класс ТекстовыйПроцессор расширен путем явного указания его версии и автора.

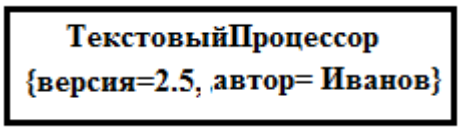

Рисунок 18 – Расширение класса

Стереотип (stereotype) расширяет словарь языка, позволяет создавать новые виды строительных блоков, производные от существующих и учитывающие специфику новой проблемы. Элемент со стереотипом является вариацией существующего элемента, имеющей такую же форму, но отличающуюся по сути. У него могут быть дополнительные ограничения и теговые величины, а также другое визуальное представление. Он иначе обрабатывается при генерации программного кода. Отображают стереотип как имя, указываемое в двойных угловых скобках (или в угловых кавычках).

Примеры элементов со стереотипами приведены на рисунке 19. Стереотип «exception» говорит о том, что класс ПотеряЗначимости теперь рассматривается как специальный класс, которому, положим, разрешается только генерация и обработка сигналов исключений. Особые возможности метакласса получил

класс ЭлементМодели. Кроме того, здесь показано применение стереотипа «call» к отношению зависимости (у него появился новый смысл).

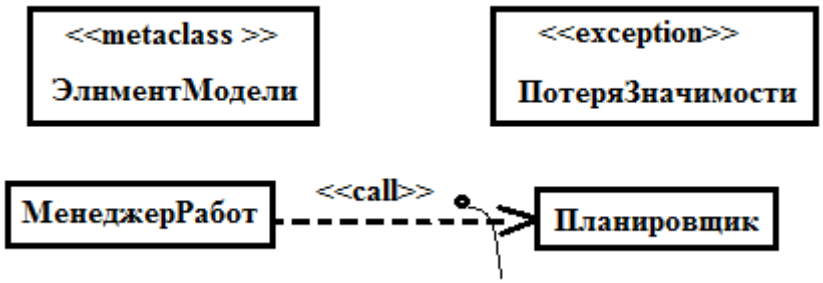

Рисунок 19 – Стереотипы

Таким образом, механизмы расширения позволяют адаптировать UML под нужды конкретных проектов и под новые программные технологии. Возможно добавление новых строительных блоков, модификация спецификаций существующих блоков и даже изменение их семантики. Конечно, очень важно обеспечить контролируемое введение расширений.