Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный университет имени П.М. Машерова»
Кафедра прикладной математики и механики

**О.Г. Казанцева, В.В. Новый,
С.А. Ермоченко**

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Методические рекомендации

Витебск
ВГУ имени П.М. Машерова
2015

УДК 004.2(075.8)
ББК 32.971.35-02я73
К14

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 1 от 23.10.2015 г.

Авторы: старшие преподаватели кафедры прикладной математики и механики ВГУ имени П.М. Машерова **О.Г. Казанцева, В.В. Новый;** заведующий кафедрой прикладной математики и механики ВГУ имени П.М. Машерова, кандидат физико-математических наук **С.А. Ермоченко**

Р е ц е н з е н т :
доцент кафедры автоматизации технологических процессов и производства УО «ВГТУ», кандидат технических наук, доцент *В.Е. Казаков*

Казанцева, О.Г.

К14 Проектирование информационных систем : методические рекомендации / О.Г. Казанцева, В.В. Новый, С.А. Ермоченко. – Витебск : ВГУ имени П.М. Машерова, 2015. – 43 с.

В методических рекомендациях изложены общие подходы при проектировании информационных систем различного уровня. Даны краткие теоретические сведения о методологиях проектирования информационных систем, в частности о методологии RUP. Приведен список вариантов предметных областей (с кратким описанием требований к каждой из них) для использования в качестве задания студентам на выполнение итогового проекта.

УДК 004.2(075.8)
ББК 32.971.35-02я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данных методических рекомендациях приведены краткие теоретические сведения по проектированию информационных систем, в основном их программного обеспечения. Материал разбит на 2 раздела.

В 1 разделе собраны сведения о проектировании, о жизненном цикле разработки программного обеспечения и о методологии проектирования. В том числе приведены сведения о методологии RUP и унифицированном языке моделирования UML. Эта методология в последнее время практически не используется в чистом виде, так как она предполагает большое количество документации и длительный период проектирования. Более популярны, особенно в проектах, над которыми работают небольшие команды, гибкие методологии управления процессом разработки программного обеспечения и проектирования информационных систем. Гибкие методологии предполагают работу команды разработчиков в тесном взаимодействии с заказчиками и максимально короткие циклы, на которые разбивается весь процесс разработки. В таком случае большое количество документации не требуется и даже наоборот, мешает процессу разработки, в том числе и увеличивая стоимость проекта. Но полностью отказаться от документации на проектах тоже не возможно, поэтому частично гибкие методологии заимствуют виды документов и UML-диаграммы, использующиеся в RUP методологии. Таким образом, сведения, приводимые в данном разделе, будут полезны и в контексте современных тенденций управления проектами и методологий проектирования.

Во 2 разделе приводится список вариантов предметных областей, которые могут использоваться в качестве задания на индивидуальные проекты для студентов по различным дисциплинам, изучающим методологии проектирования или технологии разработки программного обеспечения информационных систем.

Материал соответствует отдельным темам рабочих программ курсов: «Избранные главы информатики» (специальность «Прикладная математика»), «Проектирование программных систем», «Web-программирование», «Технологии разработки информационных систем», «Шаблоны проектирования» (специальность «Прикладная информатика»), «Прикладная информатика», «Объектно-ориентированные технологии программирования и стандарты проектирования» (специальность «Программное обеспечение информационных технологий»).

1. ПРОЕКТИРОВАНИЕ И ЯЗЫК МОДЕЛИРОВАНИЯ UML

1.1. Понятие информационной системы

Информационные системы – это компьютерные системы, ориентированные на обработку данных о некоторой предметной области. Существуют различные виды информационных систем, разработка которых ведется с использованием одной из двух моделей.

1. Создание лишь программного обеспечения для ***типовой*** информационной системы и дальнейшая его продажа различным пользователям.
2. Разработка информационной системы ***по заказу конкретного клиента*** с дальнейшим развертыванием информационной системы и её поддержкой.

В данных рекомендациях основное внимание уделяется второй модели разработки информационной системы, хотя, по большому счету, с точки зрения разработчика все равно, какая модель используется. Данный выбор в пределах этих рекомендаций обусловлен более простым выявлением и анализом функциональных требований, так как пользователь для разработчика не абстрактен.

Информационная система – это совокупность нескольких компонент.

1. Информация.
2. Программное обеспечение.
3. Аппаратное обеспечение.
4. Персонал.

Информация, которая обрабатывается в системе, является самым значимым компонентом с точки зрения заказчика. Именно ради обработки информации и разрабатывается сама система. Хотя с точки зрения разработчика не важно, какая информация будет обрабатываться в системе (а зачастую, с точки зрения соблюдения конфиденциальности данных, разработчик и не будет иметь непосредственного доступа к реальным данным, обрабатываемым информационной системой). Но, тем не менее, необходимо разработать такую систему, которая не будет допускать потери значимой информации или случайной модификации. Такую проблему решают, как правило, с применением специального класса программного обеспечения – систем управления базами данных (СУБД). Но и в таком случае от разработчика требуется понимание принципов обработки данных в СУБД. Так, например, необходимо предусматривать возможность обновления программного обеспечения информационной системы после его доработки разработчиками без ущерба для реальных данных, уже имеющихся в базе данных заказчика.

Программное обеспечение, необходимое для обработки информации, может использоваться как стандартное (программное обеспечение общего назначения – операционные системы, сервера баз данных, сервера приложений, клиентское программное обеспечение, например, браузеры), так и специализированное программное обеспечение, разрабатываемое специально для информационной системы (специализированные серверные и клиентские приложения, написанные, например, на C, C++, C#, Java; скрипты, выполняемые сервером приложений, например PHP-скрипты или Java-сервлеты; скрипты, выполняемые стандартным клиентским программным обеспечением, например JavaScript, Java-апплеты, Flash-ролики);

Аппаратное обеспечение – это оборудование, имеющееся у заказчика, или специально подобранное, собранное или созданное разработчиками, обеспечивающее хранение информации и его обработку с помощью программного обеспечения информационной системы, в том числе и её передачу между компонентами системы (серверное оборудование, персональные компьютеры, мобильные устройства, коммутационное и сетевое оборудование).

Персонал, а точнее знания, умения и навыки персонала, который будет работать с информационной системой. Данный компонент выделяют как составную часть самой информационной системы, так как при разработке некоторой системы, имеющей пользовательский интерфейс, для работы с которым персоналу не хватит квалификации (использующий, например, слишком сложную профессиональную терминологию), итоговый продукт не сможет функционировать в полной мере, так как персонал не сможет вводить данные в систему и, соответственно, их обрабатывать. С точки зрения процесса разработки этот компонент важен для проектирования пользовательского интерфейса или для планирования процесса обучения пользователей работе с информационной системой.

1.2. Классификация информационных систем

Рассмотрим теперь классификацию информационных систем, так как проектирование того или иного вида систем может иметь определённые особенности.

Информационные системы можно классифицировать по различным критериям. В частности ***по архитектуре*** (см. рисунок 1.1) информационные системы делятся на виды, которые различаются способом взаимодействия программных компонент, входящих в состав информационной системы, друг с другом и с данными, обрабатываемыми внутри информационной системы.

Рисунок 1.1. Классификация информационных систем по архитектуре

Так в настольных системах все программные компоненты собраны на одной вычислительной платформе (компьютер, мобильное или другое вычислительное устройство). При этом настольные системы могут быть и многопользовательскими, например, пользователи могут работать с системой по очереди, или даже одновременно, если используется терминальная система (при этом в качестве терминала используется устройство, не являющееся отдельным вычислительным устройством, например суперкомпьютеры, к которым подключено несколько мониторов с клавиатурами).

В распределенных системах части программного обеспечения информационной системы распределены по различным вычислительным узлам, взаимодействующим между собой через компьютерную сеть. По способу взаимодействия программных компонент с обрабатываемыми данными, распределенные системы делятся на файл-серверные и клиент-серверные системы.

В файл-серверных системах различные программные компоненты взаимодействуют с данными, но не зависят от природы этих данных. Например, файл-серверные системы, построенные на базе протокола FTP, состоят из FTP-серверов, хранящих файлы, и клиентского программного обеспечения, которые являются FTP-клиентами. Такое клиентское приложение предоставляет пользователю доступ к этим данным, производит их обработку. Но при этом работа серверной части не привязана к характеру обрабатываемых данных или к структуре этих данных.

В клиент-серверных информационных системах серверное программное обеспечение (или некоторая его часть) строго привязано к конкретной предметной области и не может быть без изменений перенесено на другую предметную область. Допускается специализация на конкретной предметной области и клиентского программного обеспечения.

Также можно классифицировать информационные системы ***по количеству программных компонент***, входящих в ее состав.

1. Одноуровневые.
2. Двухуровневые.
3. Трёхуровневые.

При этом учитывается программное обеспечение, работа которого ориентирована на конкретную предметную области информационной системы.

Одноуровневые информационные системы имеют в своем составе только один программный компонент, тем не менее, такие информационные системы могут быть как настольными (наиболее часто встречающийся вариант), так и распределенными. В качестве примеров таких информационных систем можно привести специализированные настольные или мобильные приложения, которые для хранения используют:

– различные файлы и самостоятельно их обрабатывают (например, в XML-формате или CSV-формате с помощью специальной библиотеки, или в некотором собственном текстовом или бинарном формате);

– встраиваемые базы данных, обработка данных в которых производится самим приложением при помощи специализированных библиотек (например, СУБД SQLite, СУБД JavaDB);

– настольные базы данных, обработка данных в которых специализированным приложением, связь с которым информационная система осуществляет через некий известный интерфейс API (например, СУБД Microsoft Access).

Примером одноуровневой распределённой системы может являться некоторое серверное программное обеспечение, использующее для хранения данных один из вышеописанных способов. При этом доступ к этому серверу осуществляется с помощью некоторого терминального клиента, который сам не является вычислительным устройством.

Двухуровневые информационные системы имеют в своем составе два программных компонента, работа которых привязана к предметной области (то есть не может быть перенесена без изменений на другую предметную область). Такие информационные системы являются распределенными. На уровне доступа к данным используется серверное программное обеспечение (как правило, это некая серверная реляционная СУБД, поддерживающая структурированный язык запросов SQL). Работа этого программного компонента, естественно, зависит от предметной области (от структуры базы данных). На стороне клиента используется специально разработанное программное обеспечение, выполняющее визуализацию и обработку данных. Такие информационные системы соответствуют концепции «***толстого клиента***» – это специально разработанное приложение, которое не может быть использовано в других целях, но оно может использоваться и в трёхуровневых системах. Тем не менее, в двухуровневых распределённых информационных системах может быть использован только «толстый клиент».

В трёхуровневых информационных системах серверная часть разделяется на две части, взаимодействующие между собой с

использованием сетевых технологий, хотя в частности они могут функционировать на одной аппаратной платформе. При этом в качестве клиентского программного обеспечения может выступать как «толстый клиент», так и «тонкий клиент». Отличительной особенностью трёхуровневой системы является наличие сервера, отвечающего за хранение данных (например, некая серверная реляционная СУБД, поддерживающая структурированный язык запросов SQL, или другая СУБД, использующая иерархические, объектно-реляционные и документо-ориентированные базы данных), и сервера приложений (web-сервер Apache с поддержкой скриптового языка программирования PHP, web-сервер Apache Tomcat с поддержкой языка программирования Java или web-сервер Internet Information Server с поддержкой технологии Microsoft ASP .NET).

«***Тонкий клиент***» – это стандартное приложение, которое не создавалось специально для информационной системы. Как правило, такое приложения в составе информационной системы только визуализирует данные, но может и производить некоторую их обработку (например, если в качестве «тонкого клиента» используется браузер с поддержкой технологий JavaScript, Adobe Flash, JavaFX и др.).

Как правило, разделение информационной системы на уровни не только диктуется требованиями к способу взаимодействия пользователя с информационной системой, но и удобством (скоростью, стоимостью, качеством и т. д.) разработки специализированного программного обеспечения. В таком случае важным является вопрос распределения обязанностей между уровнями. Классическим при этом можно считать подход, при котором используется трёхуровневая модель с «тонким клиентом», при этом сервер баз данных отвечает лишь за хранение данных, клиент – за их визуализацию, а всю логику обработки данных обеспечивает сервер приложений. Большинство существующих серверов реляционных баз данных могут не только выполнять SQL-запросы, но и выполнять триггеры, хранимые процедуры, с помощью которых можно и обрабатывать данные. Однако эти способы трудно реализуемы, язык триггеров и хранимых процедур не так хорошо структурирован, как языки программирования высокого уровня (C++, Java, PHP и т.д.), и не так хорошо переносим на другие платформы. Всё это делает обработку данных средствами самой СУБД достаточно дорогостоящей.

В этом отношении даже функции по обеспечению целостности данных иногда возлагают на сервер приложений, не используя стандартные возможности серверов баз данных.

Также есть особенности использования «толстого клиента». И хотя такой клиент часто разрабатывается с применением объектно-ориентированного языка программирования, тем не менее, у такого решения есть ряд недостатков. Так, например, есть проблема с поддержкой

такого клиентского программного обеспечения (в случае необходимости обновить клиентское программное обеспечение или расширить круг пользователей). Такие задачи влекут за собой дополнительные затраты на сопровождение такого программного обеспечения. Поэтому в большинстве случаев более гибким и удобным для пользователей решением является использование «тонкого клиента». Даже если нужно выполнить простейшую обработку данных (сортировка и группировка данных, проверку корректности вводимых пользователем данных и т. д.), с этим чаще всего успешно справляется и обычный браузер. Но иногда все же концепция «толстого клиента» используется для снятия нагрузки на сервер.

Следующий критерий классификации информационных систем – это ***характер обработки данных***:

1. информационно-поисковые системы;
2. информационные системы обработки данных.

Отличие в этих системах заключается в сложности алгоритмов обработки информации. Например, к поисковым системам можно отнести системы хранения видео-архивов, поддерживающие возможности добавления, поиска и просмотра видеоматериалов. Но могут существовать системы, которые позволяют непосредственно с использованием возможностей программного обеспечения информационной системы производиться редактирование (монтаж, накладывание различных эффектов, преобразование видео) материала, а не просто замена одного файла измененной во внешнем приложении его обновлённой версией. Такие системы уже будут относиться к системам обработки данных. Отличительной особенностью таких систем является использование сложных математических моделей, на которых основываются алгоритмы по обработке данных. Также информационные системы обработки данных характеризуются повышенными требованиями к вычислительной мощности аппаратного обеспечения.

Можно также классифицировать информационные системы по ***масштабу*** (круг пользователей этой информационной системы):

1. Персональные – информационные системы для одного человека, при этом они не обязательно должны быть настольными.
2. Групповые – информационные системы для группы пользователей. При этом группа пользователей должна быть обозримой (например, все сотрудники некоторой компании, все студенты и преподаватели университета и т. д.).
3. Корпоративные – информационные системы с достаточно размытым кругом пользователей (например, web-приложения компании Google имеют огромное количество пользователей, но даже администраторы

серверов этой компании вряд ли смогут наверняка сказать, кто из зарегистрированных пользователей всё ещё пользуется их ресурсами, а кто уже перестал это делать).

1.3. Жизненный цикл информационных систем

Создание информационной системы – это достаточно сложный процесс, проходящий несколько этапов. Количество этапов и их назначение могут отличаться, но, как правило, выделяют пять этапов, в том или ином виде присутствующих всегда.

Термин «этап» сложился исторически, когда каждый из них следовал после окончания предыдущего. И хоть в настоящий момент этапы могут следовать нелинейно, некоторые могут идти параллельно, некоторые могут многократно повторяться и т.д., их по-прежнему принято называть «этапами».

1. Формирование и анализ требований.
2. Проектирование.
3. Разработка (программирование).
4. Тестирование.
5. Внедрение.

Первый этап – анализ. На этом этапе ведется работа с заказчиком или будущими пользователями информационной системы. Цель данного этапа – выявить требования к продукту (что он должен и чего не должен делать) и формализовать эти требования для разработчиков, чтобы постановка задачи была ясна программистам, не являющимся специалистами в предметной области информационной системы.

Второй этап – проектирование. На этом этапе выбираются языки программирования и среды разработки, а главное – строятся необходимые модели (в первую очередь модель предметной области), обеспечивающие корректную обработку данных. Далее проектируется структура информационной системы на различных уровнях. Цель данного этапа – создать такой каркас системы, который удовлетворит сформированным требованиям (производительность, масштабируемость, простота модификации и т.п.).

Третий этап – разработка. На этом этапе создается с использованием различных структур данных модель предметной области, а также с применением выбранных языков программирования реализуются алгоритмы, обрабатывающие данные информационной системы.

Четвертый этап – тестирование. На этом этапе оценивается корректность работы системы, выявляются и исправляются дефекты (работа системы, не совпадающая со сформулированными требованиями). Если информационная система использует сложные математические

модели для анализа и обработки данных, проверяется адекватность этих моделей. Проверяются используемые алгоритмы на устойчивость их работы на самых разных наборах входных данных. Цель этапа – убедиться, что разработанная система имеет приемлемое качество.

Пятый этап – внедрение. На этом этапе осуществляется ввод в эксплуатацию разработанного продукта, обучение (при необходимости) пользователей работе с системой, исправление обнаруженных ошибок, пропущенных при тестировании, сбор сведений о необходимых улучшениях в следующих версиях программного обеспечения.

Как уже было сказано, различные этапы жизненного цикла не обязательно идут линейно один за другим. Порядок следования этапов, их повторяемость, задачи, решаемые при очередном повторении некоторого этапа, определяются выбранной разработчиками моделью жизненного цикла информационной системы. Таких моделей может существовать большое количество. По большому счету модель жизненного цикла разработки программного обеспечения в каждом проекте уникальна. Тем не мене принято выделять несколько типичных моделей, комбинацией которых можно получить оптимальную для конкретного проекта модель.

Исторически первой моделью можно назвать ***каскадную*** модель (см. рисунок 1.2). Именно такая модель породила термин «этап». Согласно этой модели (в её первоначальном виде), все этапы должны идти в строгой последовательности друг за другом и проходить только один раз (как вода, проходящая каскадный водопад, проходит каждый порог, и только один раз). На практике такая модель стала практически неосуществима, так как на любом из этапов может выявиться ошибка, допущенная на предыдущем этапе, что парализует дальнейшую разработку проекта.

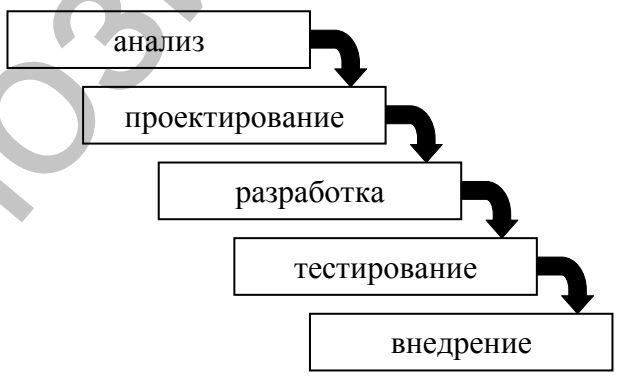

Рисунок 1.2. Каскадная модель жизненного цикла информационной системы.

Так как на любом этапе жизненного цикла потенциально возможны ошибки, влияющие на последующие этапы, при выявлении таких ошибок целесообразно вернуться на тот этап, на котором была допущена ошибка, и исправить её. Не всегда, правда, удается определить, на каком этапе была допущена ошибка, поэтому иногда используют возврат на предыдущий

этап, если там ошибку исправить не удалось, то на предшествующий этому этап и так далее (см. рисунок 1.3). Такая модель позволяет учесть самые разные факторы, влияющие на процесс разработки, и в пределе может дать идеальный результат. Но проблема использования такой модели заключается в непрогнозируемых материальных, людских и временных затратах. То есть с применением этой модели можно создать идеальную информационную систему, но остаётся нерешённым вопрос, сколько труда, времени и денег для этого понадобится.

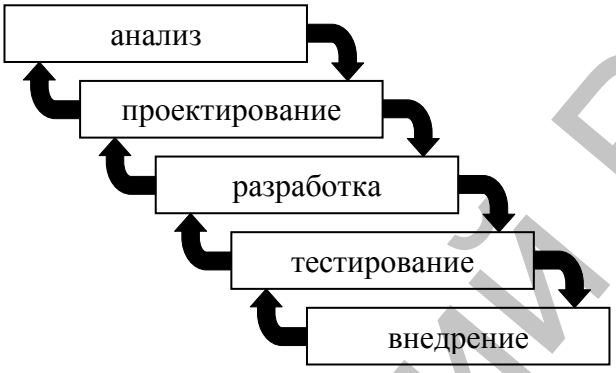

Рисунок 1.3. Модифицированная каскадная модель жизненного цикла

Одним из способов решения такой проблемы является разбиение информационной системы на ряд подзадач, решение каждой из таких задач представляется как разработка «мини информационной системы», для которой применяется своя модель жизненного цикла.

Такая модель получила название ***итерационная***. Итерацией в такой модели называется подзадача, отдельно решаемая с применением каждого из этапов. По завершении одной итерации, на которой были пройдены все этапы, начинается следующая итерация, на которой вся последовательность этапов повторяется снова.

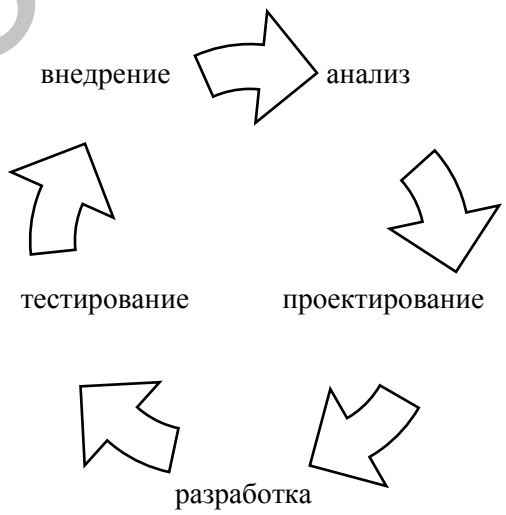

Рисунок 1.4. Модифицированная каскадная модель жизненного цикла

При этом существует большое количество вариаций данной модели. Например, из циклического повторения этапов может исключаться анализ, проводимый только один раз вначале проекта, или внедрение, проводимое один раз в конце проекта.

Существует вариант итерационной модели, при котором каждая итерация может также «зацикливаться», то есть после завершения основной работы по итерации продолжается улучшение функционала до приемлемого уровня качества (при этом параллельно могут разрабатываться другие итерации).

Применение итерационной модели при минимально возможной длительности итерации (1–2 недели) при уменьшении количества документации, более плотным сотрудничеством непосредственно с заказчиком, включающем более частое предъявление заказчику промежуточных результатов разработки, стало очень популярным в так называемых гибких методологиях управления проектами.

Дальнейшим развитием итерационной модели является ***спиралевидная*** модель, которая отличается от итерационной модели ориентацией на управление рисками. В процессе создания информационной системы существуют следующие риски:

– дефицит специалистов;
– нереалистичные сроки и бюджет;
– несоответствие спецификации;
– перфекционизм;
– постоянные изменения;
– недостаточная производительность;
– разрыв в квалификации специалистов.

При планировании сроков итерации и объёмов задач, которые должны будут решаться на итерации, менеджер проекта оценивает все эти риски и оперативно реагирует на такие, которые превышают некий допустимый порог. Рассмотрим более подробно эти риски.

Дефицит специалистов может возникать в тех ситуациях, когда объём задач, решаемых на текущей итерации, требует большого количества специалистов определённой области, в то время как количество специалистов этой области, задействованное на проекте, меньше требуемого количества. В такой ситуации, менеджеру необходимо перераспределять такие задачи между текущей и последующими итерациями для того, чтобы их можно было решить силами имеющихся специалистов. К этой же области можно отнести и другую проблему планирования задач в рамках проекта, когда на некоторые итерации выставляются достаточно простые задачи, которые вынуждены решать специалисты высокой квалификации. А через несколько итераций скапливается ряд сложных задач, которые распределить равномерно на оставшиеся итерации уже невозможно таким образом, чтобы решить их

силами имеющихся высококвалифицированных специалистов. В такой ситуации приходится или привлекать дополнительных специалистов, что приводит к превышению бюджета проекта, либо отодвигать сроки выполнения проекта. Обе ситуации могут быть неприемлемыми для заказчика. Такая ситуация как раз и создаёт риск нереалистичных сроков и бюджета проекта.

Риск несоответствия спецификации (документированные требования) возрастает с ростом сроков работы над итерациями и сроков представления промежуточных результатов заказчику. Дело в том, что никакая спецификация не отражает в полной мере ожидания заказчика. Поэтому возникает разрыв между пониманием требований заказчиком и разработчиком. Следствием такого разрыва является выполнение некоторой итерации (или нескольких итераций), которая формально соответствует спецификации, но реально не устраивает заказчика. В таком случае часть проделанной работы заказчиком должна оплачиваться, но пользы никакой не приносит. Для снижения такого риска рекомендуется как можно чаще демонстрировать промежуточные результаты работы заказчику, что позволит более конкретно понять и детализировать спецификацию и улучшить степень соответствия информационной системы этой спецификации.

Риск перфекционизма или постоянных изменений возникает в случае, когда зацикливаются один или два этапа (в первом случае это разработка и тестирование, во втором случае это анализ требований). При этом некоторая работа на этих этапах проводится, но результат этой работы несоизмерим с затратами. Например, после нескольких итераций по выявлению ошибок на этапе тестирования и их исправлению на этапе разработки, может складываться ситуация, когда новые незначительные ошибки выявляются, программисты тратят время на их исправление, а общее качество проекта при этом увеличивается на десятые доли процентов. При анализе требований аналитик может тратить много времени на согласование требований из-за того, что заказчик никак не может определиться, как именно должен работать тот или иной функционал. Но при этом из времени, отводимого на выполнение всего проекта, остаётся всё меньше и меньше на реальную работу. В таком случае необходимо директивно прерывать такие зацикливания, либо отодвигая такую работу на более поздние итерации (в случае риска перфекционизма), либо (в случае постоянных изменений) принимая хотя бы часть требований, которые не в полной мере устраивают заказчика, но дают возможность быстрее техническим специалистам приступить к работе, быстрее создать некоторую промежуточную версию, которую можно продемонстрировать заказчику, что, в том числе, может помочь заказчику лучше понять, что ему необходимо.

Риск недостаточной производительности возникает, когда специалисты долгое время задействованы на решении очень похожих задач, что приводит к рутинным действиям. Собственно, итогом такой ситуации и является снижение производительности труда специалистов. В общем случае факторов, приводящих к снижению производительности, может быть и больше. Решением такой проблемы может стать переключение специалистов на новые виды задач в рамках текущего проекта, или даже, если имеется возможность и необходимость, переключение специалиста на другой проект.

Риск разрыва в квалификации специалистов может возникать в случае, когда на одном этапе работают специалисты одной квалификации, а на следующем этапе – специалисты с квалификацией, которая существенно ниже квалификации специалистов предыдущего этапа. В таком случае есть риск, что работа на втором этапе будет выполняться более длительное время и с меньшим качеством, так как специалистам на этом этапе понадобится больше времени на то, чтобы понять и осознать результаты предыдущего этапа.

1.4. Формирование и анализ требований

На этапе формирования и анализа требования решаются следующие основные задачи:

– сбор требований;

– анализ требований;

– документирование требований.

За решение этих задач отвечают следующие специалисты:

– бизнес-аналитик (business analyst – BA) – отвечает за сбор и анализ требований, является экспертов в некоторой предметной области, может не иметь специальных знаний в области информационных технологий и программирования, но имеет опыт формулировки требований понятным для технических специалистов языком;

– дизайнер (designer) – занимается проектированием пользовательского интерфейса, является экспертом в области создания эффективных и удобных интерфейсов, может подключаться к работе и на этапе проектирования, или даже на этапе разработки, но чем раньше начнётся создание интерфейса, тем лучше будут согласованы требования, так как визуальный интерфейс легче воспринимается как заказчиком, так и разработчиками.

Примечание: создание интерфейса приложения, позволяющего заказчику протестировать работу с ним и оценить удобство использования, называется ***прототипированием****. Например, прототипом может являться набор свёрстанных HTML-страниц для web-приложения,*

или настольное или мобильное приложение, содержащее визуальные формы, но не реализующее никакой логики. Также к понятию прототипа можно отнести обычные рисунки в одном из графических форматов, изображающие внешний вид приложения. Прототипами могут служить и схематические рисунки, демонстрирующие лишь концепцию пользовательского интерфейса (например, расположение элементов интерфейса и их состав). В любом случае, способ и объём прототипирования определяются исходя из специфики проекта. Например, для рекламных сайтов определяющим является именно внешний вид, а не его функциональные возможности. Соответственно, и прототип будет ориентирован, прежде всего, на эффектный внешний вид. А для бизнес-приложения в области бухгалтерского учёта важнее функционал, поэтому на первых этапах пользовательский интерфейс достаточно спроектировать схематически.

– технический писатель (technical writer) – занимается непосредственным оформлением документов, описывающих требования, следит за соблюдением формальных требований к таким документам.

При сборе требований основными путями выявления требований являются следующие.

– Нормативные документы. В случае, если функционал информационной системы привязан к неким законодательным актам или локальным положениям, приказам и регламентам организации-заказчика, в таком случае это основной способ выявления требований, обладающий высокой степенью формализации и документированности.

– Возможности аналогов. Когда разрабатываемая информационная система имеет некий аналог, то большую часть функционала можно не описывать в требованиях, а ограничиться лишь требуемыми отличиями и нюансами.

– Ожидания пользователей. С одной стороны – самый естественный способ сбора требований, но и самый трудоёмкий и неоднозначный. Как правило – информационная система – это многопользовательская сложная система с распределением различных ролей. В таком случае практически невозможно опросить всех пользователей, которые будут работать с информационной системой, и уж тем более учесть все их пожелания. Поэтому такой способ выявления и сбора требований сопряжён с постоянным поиском компромиссов и риском постоянных изменений требований к проекту. Однако вообще не учитывать мнение пользователей тоже не правильный подход. Как правило, бизнес-аналитик должен найти ту золотую середину, которая позволит ему собрать все требования, но при этом не отвлекать потенциальных пользователей на постоянные обсуждения, так как это, всё-таки, не основной их вид деятельности.

При анализе ожиданий пользователей используют различные формы взаимодействия. Так, например, интервью может использоваться для

беседы с пользователем один на один. Часто применяется для обсуждения узкопрофильных требований, за работу с которыми отвечает конкретный уполномоченный представитель заказчика. Опросы или анкетирование применяются для того, чтобы выяснить некоторые нюансы требований (обсуждение различных вариантов) среди групп пользователей. Например, для выявления требования к информационной системе, отслеживающей успеваемость студентов некого учреждения высшего образования, нюансы требований для модулей, с которыми будут работать заведующие кафедрой, можно обсудить в форме опроса каждого заведующего, выяснив, какие нюансы есть в работе каждого из них. А вот нюансы требований для модуля, с которым будут работать непосредственно студенты, можно выяснить с помощью анкетирования большей части студентов. Для решения более сложных вопросов, касающихся требований, можно проводить семинары (для больших групп пользователей) или мозговые штурмы (для небольших групп пользователей) совместно с бизнес-аналитиком, дизайнером или другими техническими специалистами со стороны разработчика. Активно к работе с требованиями могут привлекаться технические специалисты со стороны заказчика (если таковые имеются).

При анализе собранных требований рассматриваются различные характеристики требований:

– завершённость (требование полностью, т.е. в полном объёме описано в одном месте);

– последовательность (непротиворечивость с другими требованиями);

– атомарность (требование не может быть разбито на более мелкие требования без потери завершённости);

– актуальность (требование не устарело с течением времени);

– выполнимость (требование может быть реализовано в пределах проекта, т.е. в пределах отведённого бюджета и временных рамок);

– недвусмысленность (формулировка требования не позволяет двоякого прочтения);

– проверяемость (то есть выполнение требование может быть проверено каким-либо из способов, например: непосредственный осмотр работы готового функционала; демонстрация работы в специально созданных условиях; документально подтверждённые результаты тестирования с указанием процента функционала, покрытого тестированием, процента успешно пройденных критичных, средне-критичных и некритичных тестов; а также анализ характеристик информационной системы).

Для пояснения характеристик требований приведём некоторые примеры соблюдения и несоблюдения характеристик (см. таблицу 1.1).

Таблица 1.1. Примеры требований

Характеристика	**Пример требования, не обладающего характеристикой**	**Пример требования, обладающего характеристикой**
Завершённость	*Требование #123* Список книг отображается в таблице с заголовками: – заглавие – автор – название издательства – год издания – количество страниц	*Требование #123* Список книг отображается в таблице с заголовками: – заглавие – ***фамилия и инициалы автора*** – название издательства – год издания – количество страниц
Последовательность	*Требование #124* Таблица со списком книг по умолчанию должна быть отсортирована по столбцу «жанр»	*Требование #123* Список книг отображается в таблице с заголовками: – ***название жанра*** – заглавие – фамилия и инициалы автора – название издательства – год издания – количество страниц
Атомарность	*Требование #123* Список книг отображается в таблице с заголовками: – название жанра – заглавие – фамилия и инициалы автора – название издательства – год издания – количество страниц	*Требование #123* Список книг отображается в таблице с заголовками: – название жанра – заглавие – фамилия и инициалы автора – название издательства – год издания – количество страниц
	Таблица позволяет произвести сортировку списка по любому столбцу	*Требование #125* ***Таблица со списком книг из требования #123 позволяет произвести сортировку списка по любому столбцу***
Недвусмысленность	*Требование #234* На кнопке «Сохранить» должна отображаться красивая иконка	*Требование #234* На кнопке «Сохранить» должна отображаться ***иконка из файла "/img/save-icon.png"***
	Требование #235 Таблицы с большим количеством строк разбиваются на несколько страниц	*Требование #235* Таблицы с ***количеством строк более 50*** разбиваются на несколько страниц
	Требование #236 В случае длительного ожидания загрузки данных необходимо отобразить индикатор загрузки	*Требование #236* В случае ожидания загрузки данных ***дольше 50 мс*** необходимо отобразить индикатор загрузки

При документировании требований могут создаваться различные виды документов, которые могут дополняться визуальными диаграммами, созданными в соответствии с унифицированным языком моделирования UML (Unified Modeling Language). На постсоветском пространстве стандартом на оформление проектной документации является техническое задание (регламентируется одним из документов ГОСТ 19.201-78, ГОСТ 34.602-89, ГОСТ 25.123-82 и др.). В западноевропейской и американской традиции вместо единого документа оформляются различные документы, которые содержат различные виды требований:

– бизнес-требование;
– пользовательские требования;
– функциональные требования;
– нефункциональные требования.

Бизнес-требование описывает общую цель создания информационной системы и глобальные задачи, которые она должна решать. Описывается в документе «Vision and Scope».

Пользовательские требования описывают те действия, которые могут сделать пользователи с различными ролями, в информационной системе. Пользовательские требования отвечают на вопрос «***что может сделать пользователь?***». Используются для описания общей концепции информационной системы. Также могут применяться как способ структуризации функциональных требований. Документируются с помощью UML-диаграмм вариантов использования (Use Case Diagram). Также такие диаграммы называются диаграммами прецедентов. Используются как в документе «Vision and Scope», так и в других документах.

Функциональные требования подробно описывают, каким образом должны функционировать те или иные части информационной системы. Именно в функциональных требованиях описываются все тонкости бизнес-логики приложения. Функциональные требования отвечают на вопрос «***как будет работать система?***». Основной вид требования, которым должны руководствоваться разработчики. Описываются в виде текста и могут сопровождаться различной графикой. В том числе для иллюстрации функциональных требований активно используется прототипирование. Описываются в документе «System Requirements Specification» или в специальном документе «Functional Requirements».

Нефункциональные требования описывают различные технические требования к информационной системе. Сложность выявления таких требований заключается в том, что заказчик, как правило, фокусируется лишь на функциональных требованиях, и может не представлять, какие нефункциональные требования критичны для информационной системы.

Примерами нефункциональных требований могут быть требования к дизайну и удобству использования, требования к безопасности и надёжности, требования к производительности и др.

Рассмотрим теперь некоторые дополнительные характеристики требований, важные для процесса планирования сроков и бюджета проекта.

– Важность (приоритетность). Характеристика показывает, насколько критичным для заказчика является выполнение этого требования. Такая характеристика позволяет выбирать для реализации, прежде всего, самые критичные требования, что помогает концентрировать основные силы разработчиков на первоочередных задачах информационной системы.

– Сложность реализации (в человеко-часах). Характеристика позволяет планировать время работы над реализацией информационной системы. Также в сложности реализации может учитываться требуемая квалификация специалиста, который может реализовать требование. Часто сложность трудно оценить для всего требования, так как в будущем для реализации этого требования различным разработчикам будут ставиться различные задачи в рамках этого требования. В таком случае сложность оценивается для каждой задачи отдельно, с учётом квалификации разработчиков.

– Устойчивость (вероятность изменения в будущем). Характеристику трудно измерить некоторым конкретным числом. Как правило, используют одно из значений: высокая устойчивость (требование вряд ли изменится в будущем), средняя устойчивость (изменения возможны), низкая устойчивость (изменения очень вероятны).

1.5. Проектирование и язык моделирования UML

На этапе проектирования решаются следующие основные задачи.

– Проектирование модели предметной области. При этом разрабатываются структуры данных, которые будут хранить информацию о предметной области, структурируя её понятным для разработчиков способом. Для объектно-ориентированного проектирования это, прежде все, разработка структуры классов, поля которых будут хранить необходимую информацию, и определение взаимосвязей между этими классами. Также при решении этой задачи проектируется структура постоянного хранилища данных. В подавляющем количестве случаев для этого используется одна из систем управления базами данных. В большинстве случаев – это некая серверная система управления реляционными базами данных, поддерживающая структурированный язык запросов SQL (Oracle, Microsoft SQL Server, PostgreSQL, MySQL и т. д.). В последнем случае под проектированием структуры постоянного

хранилища понимается разработка ER-диаграммы базы данных и создание схемы базы данных (создание таблиц, ключей и других ограничений).

– Проектирование аппаратного обеспечения. При решении этой задачи осуществляется выбор компонент вычислительной системы, на базе которой будет разворачиваться информационная система. Прежде всего, это актуально для распределённых высоконагруженных информационных систем, которые способны одновременного обрабатывать большое число клиентских запросов. Для таких систем необходимо подобрать необходимое серверное оборудование, подходящее под планируемую нагрузку. А также спроектировать компьютерную сеть, объединяющую сервера в единую вычислительную систему.

– Проектирование программного обеспечения. При решении этой задачи проектируется набор модулей и способы взаимодействия между собой. При использовании объектно-ориентированного проектирования разрабатываются абстрактные классы и интерфейсы, обеспечивающие функционирование программного обеспечения на верхнем уровне абстракций. Эта часть проектирования называется проектированием архитектуры программного обеспечения. Также на данном этапе может проектироваться и внутреннее устройство компонентов программного обеспечения, но чаще всего это выполняется уже на этапе разработки. Также на данном этапе продолжается (или начинается) проектирование пользовательского интерфейса приложения.

За решение перечисленных задач на этапе проектирования отвечают:

– бизнес-архитектор (Business Architect), который отвечает за проектирование так называемых бизнес-процессов, то есть основной бизнес-логики функционирования информационной системы (данный специалист должен быть, в первую очередь, экспертом в области разработки программного обеспечения, но так же и иметь опыт работы с данной предметной областью, именно он отвечает за её моделирование);

– системный архитектор (System Architect), который отвечает за проектирование аппаратного и программного обеспечения (у таких специалистов может быть своя узкая специализация в области построения и администрирования компьютерных сетей, проектирования распределённых приложений, в области объектно-ориентированного проектирования и т. д.);

– администратор баз данных (DataBase Administrator – DBA), специалист в области использования баз данных, в первую очередь реляционных баз данных с поддержкой SQL-запросов (отвечает за проектирование схемы базы данных, её оптимизации для специфики разрабатываемой информационной системы и т. п.);

– ведущий разработчик (Lead Developer), в основном работает на этапе разработки (программирования), но так как на этапе проектирования, как правило, программное обеспечение не проектируется в полном объёме,

проектируются лишь общие модули и интерфейсы их взаимодействия, то подробности проектирования и реализации модулей прорабатываются на следующем этапе, где за такие работы отвечает, прежде всего, ведущий разработчик.

Основным инструментом на этапе проектирования программного обеспечения является унифицированный язык моделирования (Unified Modeling Language – UML). Этот язык состоит из различных графических диаграмм. Графическая форма языка позволяет быстрее создавать проектные решения и быстро и интуитивно их воспринимать. Рассмотрим наиболее популярные виды UML-диаграмм, применяющиеся при проектировании информационных систем.

1.6. Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case Diagram, или диаграмма прецедентов) отражает отношения между ***актёрами*** и ***вариантами использования*** (прецедентами).

Прецедент – это некоторая возможность моделируемой системы, благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Используется для спецификации пользовательского требования к приложению. Показывает, что можно сделать, но не как. Каждый вариант использования представляется на диаграмме в виде овала, в который вписывается, что может сделать пользователь. Примеры вариантов использования приведены на рисунке ниже.

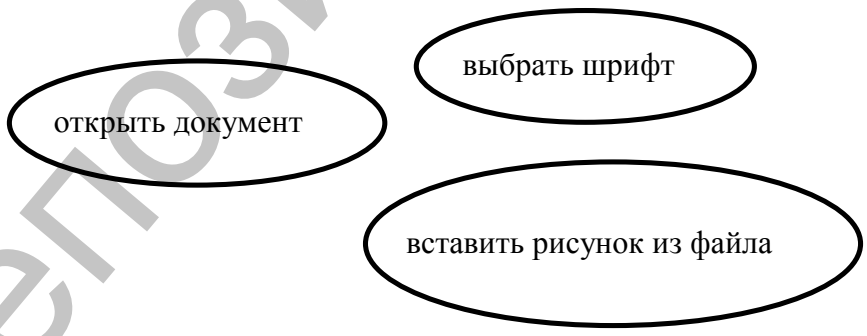

Рисунок 1.5. Примеры вариантов использования

Актёр – это роль, которую пользователи исполняют во время взаимодействия с вариантами использования. Однако в качестве актёров может выступать не только человек, но также аппаратное устройство, время или другая система. Примеры актёров представлены на рисунке ниже.

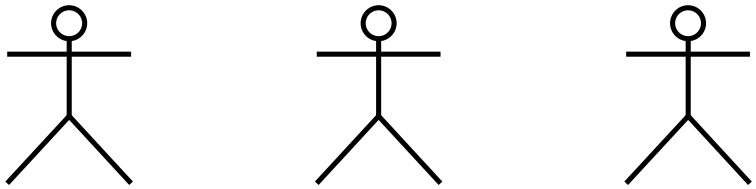

Рисунок 1.6. Примеры вариантов использования

Кроме самих актёров и вариантов использования на диаграмме используются различные виды связи между объектами диаграммы. Наиболее часто используемая связь, без которой сама диаграмма теряет смысл, это связь между актёром и вариантом использования. Такая связь называется ассоциацией (Association), и она показывает, что актёр может инициировать действие, указанное в варианте использования. Примеры таких ассоциаций показы на рисунке ниже.

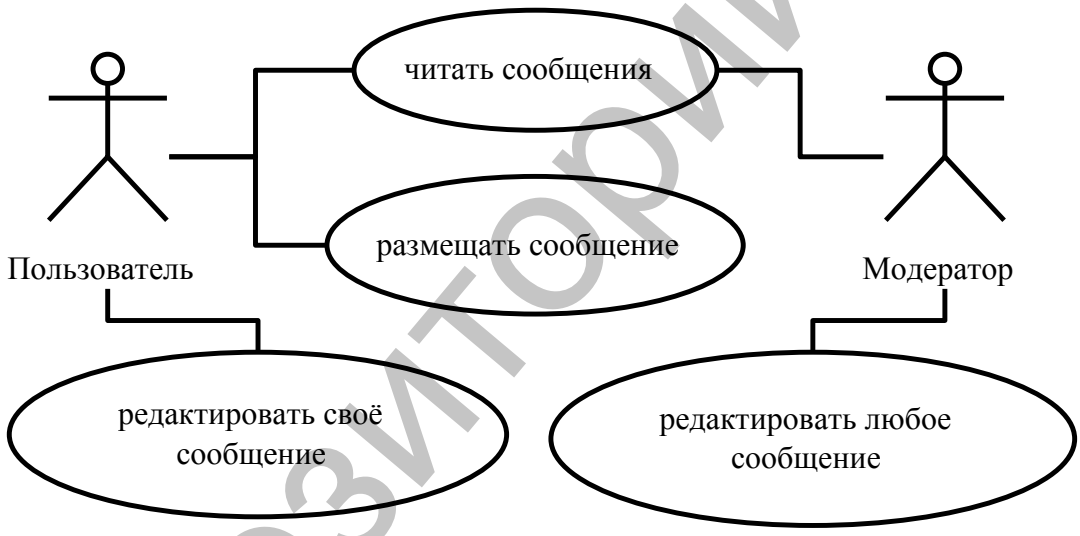

Рисунок 1.7. Пример ассоциации между актёрами и вариантами использования

Далее рассмотрим связи между актёрами. Такой вид связи называется обобщением (Generalization). Обобщение актёров (наследование) показывает, что одна из ролей является частным случаем другой роли. Фактически это обозначает, что все варианты использования, которые может инициировать более общая роль, доступны и для её наследников (но не наоборот). Пример такой связи приведён на рисунке 1.8. При этом некоторые актёры могут быть абстрактными (выделенные курсивом), что значит, что пользователя конкретно с такой ролью существовать не может.

Между вариантами использования также возможны связи, при этом различных типов:

– обобщение (Generalization);

– зависимость (Dependency), которая подразделяется на включение (Include) и расширение (Extend).

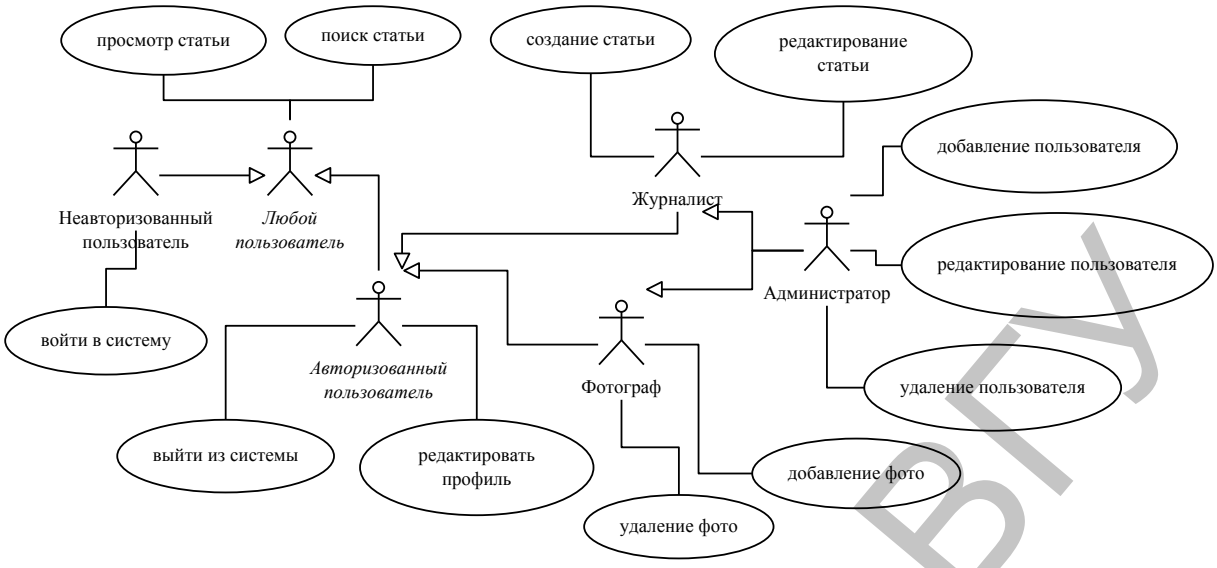

Рисунок 1.8. Пример обобщения между актёрами

Обобщение (наследование) используется, когда вариант использования, являющийся потомком, наследует поведение варианта использования, являющегося предком, дополняя его или заменяя, сохраняя общий интерфейс взаимодействия.

Обобщение между вариантами используется в том случае, когда наследник варианта использования может использоваться вместо предка. Пример таких вариантов использования смотри на рисунке ниже:

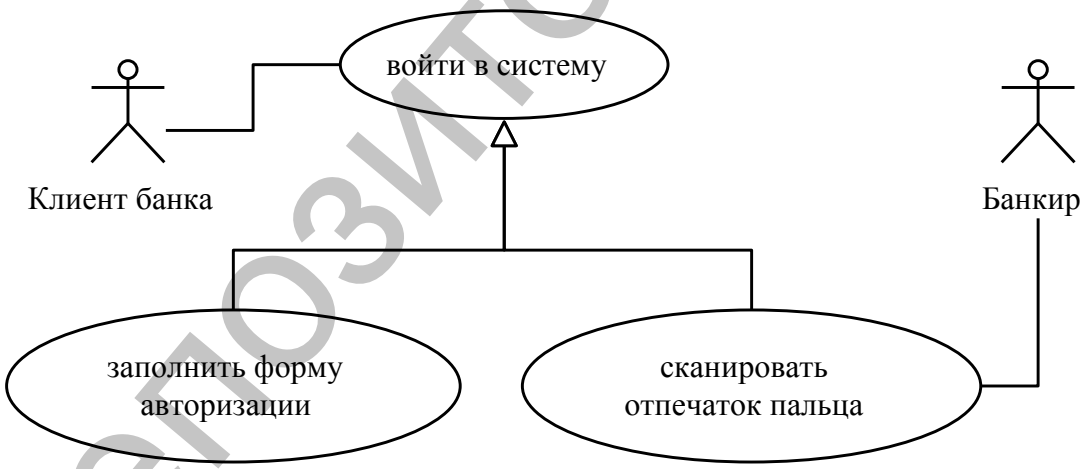

Рисунок 1.9. Пример обобщения между вариантами использования

Зависимости между вариантами использования, в общем случае, просто показывает некую зависимость. Но такая зависимость показывает непосредственную связь между выполняемыми действиями. Например, рассмотрим случай, когда для удаления некой записи, пользователю сначала необходимо открыть форму редактирования (просмотр записи), на которой он может либо исправить поля и нажать кнопку «сохранить» (редактировать запись), либо нажать кнопку «удалить» (удалить запись). В таком случае между вариантами использования «просмотр записи» и

«удаление записи» зависимости не будет. То есть последовательность действий на диаграмме вариантов использования с помощью зависимостей не отображается, как, впрочем, и никакой другой связью.

Включение вариантов использования показывает, что в некоторой точке базового варианта использования может использоваться функционал включаемого варианта использования. Пример см. на рисунке. Включаемый вариант использования не может существовать (выполняться) отдельно от базового. То есть на представленном примере ассоциация актёра с вариантом использования «просмотр фото» будет некорректным. Как правило, включаемый вариант использования описывает общую для нескольких других вариантов использования функциональность.

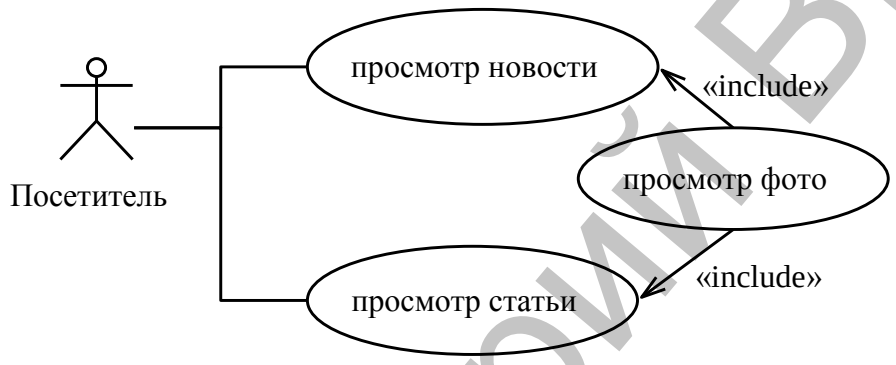

Рисунок 1.10. Пример включения вариантов использования

Расширение вариантов использования подразумевает, что базовый вариант использования неявно содержит в указанной точке функционал расширяющего. Расширяющий вариант использования передаёт своё поведение базовому (возможно при каком-то условии). В отличие от включения, расширяющий вариант использования имеет самодостаточный смысл и может отдельно выполняться актёром, этим он похож на обобщение. Но, в отличие от обобщения, выполнение актёров базового варианта использования не предполагает, что вместо него может выполняться расширяющий вариант использования (только если он явно ассоциирован с актёром).

1.7. Диаграмма классов

Диаграмма классов предназначена для описания структуры программного обеспечения информационной системы на уровне классов и взаимосвязей их друг с другом. На диаграмме отображаются сами классы, их атрибуты (поля, переменные класса), их операции (методы, функции класса) и отношения (связи) между классами.

Диаграмма классов для всей информационной системы, содержащая все разрабатываемые классы, будет иметь слишком большой размер, учитывая, что некоторые информационные системы могут содержать до нескольких тысяч классов. Поэтому диаграмму классов могут создавать для решения одной из следующих задач.

– Общее концептуальное моделирование системы (как правило, на уровне интерфейсов и абстрактных классов верхнего уровня иерархии).

– Подробное моделирование для трансляции модели в программный код (как правило, для реализации конкретного требования или модуля).

– Моделирование предметной области.

Диаграмма классов может иметь статический или аналитический вид. Вначале рассмотрим аналитический вид диаграммы классов. В этом случае все классы маркируются одной из иконок, которая показывает назначение создаваемого класса. Возможные виды классов приведены в таблице 1.2.

Таблица 1.2. Виды классов на диаграмме классов

Условное обозначение	**Описание**
 граничные классы	Такие классы обеспечивают взаимодействие с внешними относительно разрабатываемой системы факторами. Например, действия пользователя, времени или внешней системы (всё, что может являться актёром в контексте диаграммы вариантов использования). При этом класс может отвечать как за обработку действия (нажатие на кнопку, входящий запрос от внешней системы и т. д.), так и за представление (визуализацию) результата, например отображение формы с компонентом, визуализирующим данные, генерацию HTML-страницы в web-системах и т. д. Фактически, именно такие классы обеспечивают взаимодействие всей информационной системы со внешней средой, поэтому они и называются граничными.
 классы-обработчики	Классы-обработчики – это внутренние классы системы, которые отвечают за реализацию бизнес-логики приложения. К таким классам относятся все классы, отвечающие: за взаимодействие с источником данных; за обработку этих данных в соответствии с алгоритмами, которые необходимо реализовать в информационной системе; за проверку корректности этих данных и т.д. Фактически, все классы, не являющиеся граничными классами или классами-сущностями, являются классами-обработчиками.
 классы-сущности	Классы-сущности – это классы, которые хранят информацию о предметной области. Такие классы не выполняют никакой обработки информации. Хранимая информация содержится в полях класса, а все методы класса лишь предоставляют доступ к этой информации (так называемые методы get-теры и set-теры). При этом методы не должны выполнять даже проверку корректности данных, так как записывать информацию в объект с применением set-теров можно не только при вводе информации пользователем системы, но и при считывании информации из постоянного хранилища. При этом в большинстве информационных систем чтение данных из хранилища (которые уже должны быть корректны), происходит гораздо чаще, чем ввод данных пользователем (которые могут некорректными). Поэтому, с точки зрения производительности, выполнять даже элементарную проверку в классах-сущностях нецелесообразно. Такие классы используются для моделирования предметной области.

Примером диаграммы классов в аналитическом виде может быть диаграмма для некоторого требования, согласно которому в некой банковской системе кассир может перевести деньги с одного банковского счёта на другой (см. рисунок 1.11).

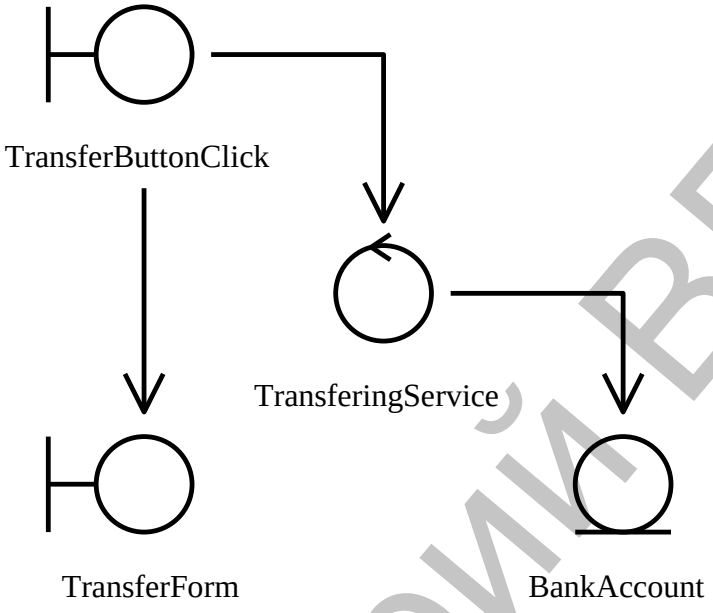

Рисунок 1.11. Аналитический вид диаграммы классов

В данном примере граничный класс TransferButtonClick обрабатывает событие нажатия на кнопку «Перевести», считывает с полей формы, которую создаёт граничный класс TransferForm, и обращается к классу-обработчику TransferingService, который изменяет состояние объектов класса-сущности BankAccount и сохраняет их в постоянное хранилище.

Рассмотрим теперь диаграмму классов в статическом виде. Класс на диаграмме в таком виде представляется прямоугольником, разделённым на три секции: имя класса, поля класса, методы класса. В качестве примера рассмотрим класс BankAccount (см. рисунок 1.12).

BankAccount
-clientName: String -balance: Integer
+getClientName(): String +setClientName(name:String) +getBalance(): Integer +setBalance(balance:Integer)

Рисунок 1.12. Класс на диаграмме классов статического вида

При отображении класса в статическом виде используются различные соглашения для отображения различных тонкостей, касающихся класса. Так, например, области видимости помечаются значками возле описания

поля и метода (знак «+» для public-членов, знак «#» для protected-членов, знак «–» для private-членов). Курсивом набираются имена абстрактных классов и интерфейсов, а также абстрактных методов. С помощью подчёркивания отображаются статические члена класса.

На диаграмме классов используются различные отношения между классами. Рассмотрим эти отношения.

Обобщение (наследование или расширение, Extend) – показывает, что один класс наследуется от другого класса. Отображается сплошной линией, заканчивающейся треугольной стрелкой с белой заливкой.

Реализация (Implement) – показывает, что класс реализует некоторый интерфейс. Отображается штриховой линией, заканчивающейся треугольной стрелкой с белой заливкой.

Ассоциация (Association) – признаком ассоциации является наличие в классе ссылки на объект другого класса. Отображается сплошной линией с обычной стрелкой.

Агрегация (Aggregation) – подвид ассоциации, моделирует связь «часть-целое». Отображается, как и ассоциация, но в начале линии ставится ромб с белой заливкой.

Композиция (Composition) – подвид агрегации, имеет привязку к времени жизни объектов. Дочерние объекты не могут существовать после уничтожения контейнера. Отображается, как и агрегация, но ромб в начале линии отображается с чёрной заливкой.

Зависимость (Dependency) – связь (всегда направленная), которая возникает в случае, если в классе есть метод, принимающий параметр-ссылку на объект другого класса, или если в классе есть метод, возвращающий ссылку на объект другого класса. Однако может существовать и неявно при создании и использовании объекта внутри некоторого метода класса. Отображается штриховой линией с обычной стрелкой.

1.8. Разработка и тестирование

На этапе разработки производится детальное проектирование каждого модуля и его реализация с применением выбранных языков программирования, библиотек и технологий.

При разработке за реализацию информационной системы отвечают разработчики (developer) и ведущие разработчики (lead developer). Ведущие разработчики выполняются детальное проектирование модулей, участвуют в их реализации и координируют действия обычных разработчиков. Обычные разработчики занимаются непосредственной реализацией программных модулей, настройкой используемого стороннего программного обеспечения и библиотек.

Для управления процессом разработки используется список заданий (Task), за каждый из которых отвечает определённый разработчик. При составлении списка заданий, как правило, каждое задание обладает

следующими характеристиками:

– описание (что нужно сделать, обычно указывается также пользовательское или функциональное требование, в рамках которого формулируется данное задание);

– планируемое время выполнение (указывается количество часов, за которое разработчик должен успеть выполнить это задание, при этом, как правило, если разработчик в это время не укладывается, так называемый overtime, то оплачивается разработчику только то время, которое было запланировано);

– статус (например: новое, принято к исполнению, завершено, отклонено, возобновлено; при этом для всех статусов, кроме новых, сохраняется информация о том, какой разработчик ответственен за дельнейшее выполнение задания);

– дата, когда задание было переведено в текущий статус (как правило, сохраняется вся история статусов заданий, чтобы можно было в течение некоторого периода отследить прогресс работы над программным обеспечением);

– предусловия (задания, которые должны быть выполнены до того, как разработчик сможет приступить к выполнению данного задания).

На этапе тестирования решаются следующие задачи:

– планирование процесса тестирования (на какой итерации и в каком объёме необходимо проводить тестирование);

– составление тестов (какой функционал подвергать тестированию и какие тесты выполнять для каждого из тестируемых функциональных требований);

– выполнение тестов (основное, что необходимо сделать тестировщику при выполнении тестов, это правильно оформить отчёт об ошибках, так называемый Bug Report, в котором обязательно указать, какие данные вводились и, вообще, какие действия предпринимались, какой результат был достигнут, и какой результат должен был быть получен согласно функциональному требованию);

На этапе тестирования основными специалистами, отвечающими за выполнение работ, являются тестировщик (Tester) и специалист по обеспечению качества (Quality Assurance – QA). Тестировщик обеспечивает выполнение тестов и составление отчёта об ошибках. Специалист по обеспечению качества планирует процесс тестирования, составляет тесты и подготавливает итоговые отчёты о результатах итерации: сколько тестов было произведено, какой процент функционала был покрыт тестами (и какого рода тестами), каков общий результат проведённых тестов (сколько процентов тестов успешно пройдено, сколько не пройдено по каждой категории требований: критичных, средне-критичных и некритичных), также обязательно выполняется сравнение с результатами тестирования предыдущих этапов, на сколько улучшился функционал, сколько ошибок было исправлено, что позволяет менеджеру проекта оценить различные риски.

2. ВАРИАНТЫ ПРЕДМЕТНЫХ ОБЛАСТЕЙ

Ниже приведены варианты заданий на разработку учебных информационных систем.

Первые 5 вариантов представляют собой системы с достаточно обширными требованиями, которые можно использовать для организации совместной работы над проектом группы студентов (3–4 человека). Можно предлагать эти задания студентам и в качестве индивидуальных проектов, упрощая или сокращая требования для этих вариантов (или разбивая один проект на 2–3 более мелких).

Оставшиеся варианты сформулированы с нечёткими требованиями и могут применяться для индивидуальных проектов, работа над которыми предполагает самостоятельное выполнение студентами этапа анализа требований.

2.1. Информационная система «Университет»

Приложение должно управлять данными о факультетах, специальностях, академических группах, студентах, кафедрах и преподавателях некоторого университета.

Данные о факультетах: название, фамилия и инициалы декана, номер телефона, номер кабинета деканата, количество обучающихся студентов, дата образования.

Данные об академических группах: факультета, название, количество человек (не может превышать на очное форме обучения 20 человек, на заочной 30 человек), специальность, курс, форма обучения (очная или заочная).

Данные о студентах: группа, фамилия, имя, отчество, пол, дата рождения, платник или бюджетник, количество не сданных экзаменов в последнюю сессию (не может превышать 2).

Данные о кафедрах: название кафедры, фамилия и инициалы заведующего, телефон, штатное расписание (количество ставок профессоров, доцентов, старших преподавателей и преподавателей, отдельно по каждой должности, при этом общее количество ставок на всех кафедрах факультета не должно превышать количество студентов дневной формы обучения, умноженное на 0,1, плюс количество студентов заочной формы обучения, умноженное на 0,15), средняя нагрузка (средняя количество часов, приходящееся на 1 ставку по каждой должности).

Данные о преподавателях: кафедра, фамилия, имя, отчество, должность (заведующий, профессор, доцент, старший преподаватель и преподаватель, при этом только у одного преподавателя на кафедре может быть указана должность заведующего), ставка (дробное число 0,25; 0,5;

0,75; 1; 1,25; 1,5, при этом у заведующего обязательно указывается ровно 1 ставка), имеет ли право на чтение лекций (если должность заведующего, профессора или доцента, то он автоматически имеет право на чтение лекции, и отнять у него это право нельзя, старшему преподавателю это право можно назначить, преподавателю ни в каком случае лекции читать нельзя), количество лекционных часов (указывается, если преподаватель имеет право на чтение лекций, при этом заведующий, доцент и профессор должен иметь не менее 100 лекционных часов на 1 ставку), количество часов практических занятий, количество часов лабораторных занятий, количество часов на консультации, количество часов на приём экзамена, количество часов на приём зачёта, количество часов на проверку контрольных работ, количество часов аудиторной нагрузки (сумма часов по лекциям, практикам, лабораторным), количество часов вне аудиторной нагрузки, общее количество часов (должно быть в указанных диапазонах для разных должностей: заведующий 600–680; профессор 675–765; доцент 750–850; старший преподаватель 863–978; преподаватель 900–1000).

2.2. Информационная система «Разработка ПО»

Приложение должно управлять данными компании по разработке программного обеспечения о заказчиках, их проектах, программистах, которые на них работают, требованиях к проектам, и выданных программистам заданий.

Данные о заказчиках: название, юридический адрес, общее количество проектов, количество завершённых проектов.

Данные о проектах: название проекта, заказчик, количество программистов, дата начала проекта, планируемая дата окончания проекта (должна быть позже даты начала), процент готовности проекта (количество часов по реализованным требованиям относительно общего количества часов по всем требованиям), потребность в программистах (количество человек, необходимых для успешного выполнения оставшихся нереализованных требований исходя из даты окончания проекта, учитывая, что один программист работает по 8 часов в день по будним дням кроме субботы и воскресенья), фактическая дата завершения, фамилия и инициалы менеджера, успешно ли завершён проект (должна быть указана дата завершения проекта, при этом отменить завершение проекта нельзя), стоимость проекта (увеличенные в 2 раза затраты на оплату труда программистов).

Данные о программистах: проект, фамилия, имя, отчество, должность, дата начала работы на проекте (должна быть не раньше даты начала проекта), дата окончания работы на проекте (должна быть позже даты начала работы на проекта и не позже даты окончания проекта), ставка оплаты часа работы, работает ли полный день, всего отработано часов (на

основании завершённых заданий), результативность (процент общего количества часов по успешно завершённым заданиям относительно количества часов по всем завершённым заданиям), прогресс (процент общего количества часов по завершённым заданиям относительно общего количества часов по всем назначенным ему заданиям), зарплата (оплата за все часы по успешно завершённым заданиям за вычетом штрафов в размере 25% от оплаты всех часов по безуспешно завершённым заданиям).

Данные требований: проект, описание, время реализации, отметка о реализации, приоритет (высокий, средний или низкий), вероятность изменения.

Данные о заданиях: программист, формулировка, планируемое количество часов, статус (принято, в процессе, успешно завершено, безуспешно завершено), дата начала, дата окончания, просрочено ли (вычисляется на основании дат начала и окончания и планируемом количестве часов).

2.3. Информационная система «Кафедры»

Бизнес требование: приложение должно управлять данными университета о факультетах, кафедрах, преподавателях, читаемых ими курсах и темах в рамках этих курсов.

Данные о факультетах: название, фамилия и инициалы декана, номер телефона, номер кабинета деканата, количество обучающихся студентов, дата образования.

Данные о кафедрах: факультет, название, фамилия и инициалы заведующего, номер телефона, номер кабинета, количестве преподавателей кафедры (при этом общее количество преподавателей всех кафедр факультета не должно превышать количество студентов, разделенное на 10), является ли выпускающей, дата образования.

Данные о преподавателях: кафедра, фамилия, имя, отчество, ученая степень, ученое звание, должность, пол, дата рождения, количество читаемых курсов.

Данные о курсах: преподаватель; название; специальность; номер курса обучения; номер семестра; количество студентов; количество лекционных часов; количество часов практических занятий; количество часов лабораторных занятий; есть ли зачёт по курсу; есть ли экзамен по курсу; количество контрольных работ; общее количество часов по курсу, которое вычисляется как сумма количества лекционных часов, практических часов, лабораторных часов, текущих консультаций (0,05 часа на каждый лекционный час), экзаменационных консультаций (2 часа), приём экзамена (0,5 часа на каждого студента при наличии экзамена), приём зачёта (0,25 часа на каждого студента при наличии зачёта), проверки контрольных работ (0,15 часа на каждого студента на каждую

контрольную работу), при этом общее количество часов в конечном итоге округляется вверх до целого числа; дата начала занятий; дата окончания занятий; необходимое количество пар для лекций, практик и лабораторных работ в неделю (рассчитывается из количества недель между датой начала и окончания занятий и общего количества соответствующих видов занятий по каждой дисциплине, при этом следует помнить, что занятие или пара – это 2 часа).

Данные о темах: дисциплина, название темы, количество лекционных часов, количество практических часов, количество часов лабораторных работ, обязательность темы (можно или нет отдать эту тему на самостоятельную работу, при этом количество часов по каждому виду занятий, выносимых на самостоятельную работу, не должно превышать 15% от общего количества часов по данному виду занятий по всей дисциплине).

2.4. Информационная система «Железная дорога»

Приложение должно управлять данными железнодорожного управления о пассажирских поездах, их вагонах, билетах в эти вагоны (свободные и проданные) и станциях по пути следования поездов.

Данные о пассажирских поездах: номер, станция отправления, станция назначения, дата и время отправления, дата и время прибытия (должна быть позже даты и времени отправления), время в пути (время прибытия и время в пути рассчитываются исходя из времени следования между станциями), количество вагонов, общее количество мест, общее количество свободных мест.

Данные о вагонах: поезд, номер вагона, тип вагона (купе, плацкарт, общий), количество мест в вагоне (для купе – 40, для плацкарта – 60, для общего – 90), количество свободных мест, фамилия и инициалы проводника, является ли вагоном для некурящих, стоимость одного билета (для общего – $ 10, для плацкарта – $ 15, для купе – $ 25, в вагоне для курящих стоимость увеличивается на 25%), общая выручка от продажи билетов.

Данные о билетах: вагон, номер места, детский ли, включается ли багаж (только если билет не детский), стоимость (для детского билета стоимость уменьшается на 50%, если билет не детский, то возможно указать наличие багажа, и при наличии багажа стоимость увеличивается на 20%).

Данные о станциях: поезд, название станции, является ли конечной станцией, порядковый номер в маршруте (0 для станции отправления), время до следующей станции по маршруту (обязательно указывается для начальной и промежуточных станций, и не должно указываться для конечной станции.

2.5. Информационная система «Поликлиника»

Приложение должно управлять данными поликлиники о специальностях врачей, самих врачах этих специальностей и талонах на приём к этим врачам (свободные и выданные).

Данные о специальности врача: название, является ли узким специалистом, количество врачей данной специальности, ставка заработной платы, общие затраты на оплату труда врачам данной специальности по поликлинике.

Данные о врачах: специальность, фамилия, имя, отчество, дата рождения, дата приёма на работу (возраст врача при приёме на работу не должен быть меньше 20 лет), номер участка (только для тех, кто не является узким специалистом, но врач, которые не является узким специалистом, обязательно должен иметь номер участка), номер кабинета, время начала приёма, время окончания приёма, количество свободных талонов на приём, общее время приёма в часах по всем выданным талонам, заработная плата (вычисляется как сумма ставки заработной платы, надбавка за стаж: 5% при стаже от 5 до 10 лет, 10% от 10 до 20 лет, 15% от 20 до 35 лет, 20% от 35 лет и выше – верхние пределы стажа не включается в границы интервалов, пенсия в размере 50% от ставки заработной платы, при этом пенсионный возраст женщин – 55 лет, мужчин – 60 лет).

Данные о талонах: врач, время начала приёма по талону, длительность приёма (время начала приёма по талону и длительность приёма должны быть согласованы со временем работы специалиста и другими талонами, общая длительность приёма всех талонов одного врача не должна превышать 80% времени, отведённого на работу врача), свободен ли талон, фамилия и инициалы пациента (обязательно заполняется, но только в случае выдачи талона).

2.6. Информационная система «Общежитие»

Приложение должно управлять информацией о корпусах общежитий, комнатах в каждом корпусе, занятых и свободных местах в комнатах. Приложение должно предоставлять возможность студентам подавать заявку на заселение в общежитие, специалистам воспитательного отдела – распределять студентов по корпусам общежитий, комендантам общежитий – распределять студентов по комнатам. При распределении студентов нужно учесть, что в одну комнату можно заселять или парней, или девушек. В двуместные номера можно заселять студенческие супружеские пары. При выселении студентов необходимо указывать причину выселения и не позволять его повторное заселение ни в один из корпусов. Необходимо предусмотреть также возможность генерации отчётов:

– о проценте иногородних студентов, заселённых в общежитие (в том числе по каждому факультету и курсу);

– о количестве свободных мест в общежитиях (в том числе по каждому корпусу и с учётом пола студентов);

– о количестве неудовлетворённых заявок от студентов (в том числе по каждому факультета и курсу).

2.7. Информационная система «Бюро кредитных историй»

Приложение должно позволять хранить информацию о физических лицах, которые могут брать кредиты в разных банках. Данные о физических лицах и их ежемесячных доходах вводит налоговая инспекция. Банки могут вводить информацию о кредитных линиях и о выдаваемых физическим лицам кредитах по этим линиям. Приложение рассчитывает для каждого физического лица планируемые ежемесячные выплаты по выданному ему кредиту. Банковские кассиры фиксируют также реальные платежи по кредитам. Система позволяет менеджерам банков, отвечающих за выдачу и оформление кредитов, просмотреть отчёт о кредитоспособности клиента. Необходимо принять во внимание его заработную плату, имевшиеся у него кредиты (по которым анализируется процент ежемесячных кредитных выплат от месячной заработной платы), имеющиеся в настоящий момент времени кредиты, а также количество просроченных ежемесячных выплат по кредитам и количество досрочно погашаемых кредитов.

2.8. Информационная система «Торговая компания»

Приложение должно позволять хранить информацию о филиалах торговой компании, их торговых представителях, клиентах, с которыми работают эти торговые представители, поставщиках товаров. Торговые представители оформляют договора с клиентами на поставку оптовых партий различных товаров. Главы филиалов анализируют продуктивность работы каждого торгового представителя (как по количеству оформленных договоров, так и по общей стоимости этих договоров). Глава компании анализирует продуктивность филиалов. Менеджеры по закупкам анализируют объёмы продаж различных товаров и закупают необходимые товары, вводя информацию о закупках.

2.9. Информационная система «Приёмная комиссия»

Приложение должно содержать сведения о факультетах, специальностях, плане набора на каждую специальность, сдаваемых экзаменах на каждую специальность и абитуриентах, подающих

документы на некоторую специальность. В приложении должна быть возможность устанавливать сроки подачи документов, назначать сроки сдачи экзаменов, срок зачисления. Абитуриента регистрирует в системе технический секретарь приёмной комиссии и сообщает ему данные для авторизации в системе. Абитуриент, входя в систему, может просматривать расписание экзаменов, после сдачи экзаменов просматривать свои результаты. Далее абитуриенты до срока зачисления могут просматривать информацию о том, проходят ли они по конкурсу на бюджет по этой специальности, и, если не проходят, перевестись на другую специальность, если по ней необходимо сдавать те же экзамены, или на внебюджет (если есть места по выбранной специальности). После окончания срока зачисления можно просмотреть список зачисленных абитуриентов, проходные балы и конкурс на каждую специальность, факультет и в целом университет (или недоборах на некоторые специальности, факультеты и университет в целом). При зачислении к участию в конкурсе допускаются только абитуриенты, успешно сдавшие все экзамены (не ниже, чем на 4 балла). При проведении конкурса суммируются баллы по каждому экзамену для каждого абитуриента, и абитуриенты зачисляются в порядке убывания этих баллов, пока не будут заполнены все места. При равном количестве баллов нескольких абитуриентов, для которых не осталось достаточного количества мест, предпочтение отдаётся абитуриенту с большим экзаменационным баллом по профильному предмету. При равных баллах по профильному предмету, предпочтение отдаётся абитуриенту с большим средним баллом аттестата. При равном среднем балле аттестатов секретарь приёмной комиссии вручную производит зачисление абитуриента, вводя причину, по которой было отдано предпочтение данным абитуриентам. Также в такой ситуации при наличии свободных мест на других специальностях можно перераспределить места между специальностями.

2.10. Информационная система «ГАИ»

Приложение должно содержать данные о водителях и выданных им водительских правах (в том числе и об изъятых полностью или на определённый срок), автомобильных средствах, совершённых правонарушениях и штрафах за эти правонарушения. В приложении должна присутствовать информация о видах правонарушениях и уровне штрафа за каждое из них. Повторное правонарушение по одной и той же статье может вести к увеличению суммы штрафа на некоторый фиксированный процент (для каждого вида правонарушения свой), изъятию водительских прав на определённый срок (для каждого правонарушения свой) или полностью. Для каждого правонарушения

фиксируется место (город, район, область) правонарушения. Приложение должно позволять выводить статистику правонарушений за определённый период по выбранному городу (городам), району (районам), области (областям), в целом по республике.

2.11. Информационная система «Прокат»

Приложение должно хранить сведения о товарах, выдаваемых в прокат, распределённых по категориям; о клиентах пункта проката; о сроках и стоимости проката. У клиентов должна быть возможность оформить предварительный заказ на прокат товара в указанные сроки (если товар будет в наличии). При возврате товара должна быть возможность фиксировать повреждения товара (необходимость в ремонте). При ремонте товара необходимо учитывать сроки ремонта для возможности предварительного заказа этого товара после окончания ремонта. В случае невозможности отремонтировать повреждённый товар, необходимо предусмотреть возможность оповещения клиента, оформившего предварительный заказ на этот товар (если товар нельзя заменить другим). Также необходимо вести учёт средств на закупку и ремонт товаров и подсчитывать прибыль от проката по каждому товару, категории и в целом всего пункта проката за указанный период.

2.12. Информационная система «Сессия»

Приложение должно обрабатывать информацию о группах, студентах, экзаменах и зачётах, которые должны сдавать студенты в сессию. В приложении должна быть возможность формировать расписание экзаменов, учитывая занятость преподавателей, принимающих экзамены, и групп студентов. Необходимо учесть при составлении расписания, что между экзаменами у одной группы должно быть не менее 3 дней на подготовку. У преподавателей должна быть возможность выставлять зачёты и оценки студентам. При этом по окончании экзамена должна формироваться статистика для экзаменационной сессии (количество студентов, получивших оценку 1, 2, 3 и т. д.). Необходимо отслеживать допуск студентов к экзамену (сдача всех зачётов). При неявке студента на экзамен секретарём деканата выставляется причина неявки (уважительная или нет). В случае неуважительной неявки (например, не допущен из-за несданных зачётов), ему автоматически выставляется 1 и студент имеет академическую задолженность. При уважительной ему выставляется пересдача преподавателем в один из дней, выставленных секретарём (период пересдачи секретарь выставляет, исходя из причины неявки,

например, если это больничный, то срок сессии продляется на количество дней, которое студент отболел). При третьей неудовлетворительной оценке студента он автоматически отчисляется из университета.

2.13. Информационная система «Интернет-магазин»

Приложение должно хранить информацию о товарах магазина, разбитых по категориям; клиентах; сделанных заказах. Клиент может оформлять заказ. Специалист по работе с клиентами согласует с клиентом по телефону и подтверждает в системе этот заказ. Работник склада формирует заказ и помечает его как готовый к отправке. Курьер доставляет заказ и подтверждает факт оплаты (указывая номер чека об оплате в системе). Клиент может отслеживать статус своего заказа. Товаровед может анализировать результаты продаж различных товаров или категорий товаров, менять стоимость товаров (но это не должно сказываться на уже оформленных заказах), добавлять количество товара (или вносить новые товары и категории), устанавливать скидки на определённые товары или категории на выбранный срок. Менеджер магазина может просматривать статистику продаж за выбранный период.

2.14. Информационная система «Сеть гостиниц»

Приложение должно содержать информацию о гостиницах (в различных городах), их номерах (количество мест в номере, вид номер: обычный, комфортный, люкс), стоимости проживания, дополнительных услугах и их стоимости (в целом за период проживания или за каждый день проживания, в зависимости от услуги). Клиент может заказать номер на указанный период (если номер свободен). Менеджер гостиницы уточняет заказ и подтверждает его, выставляя номер счёта, номер договора и срок оплаты номера. Банк подтверждает факт оплаты номера. Если номер не оплачен в срок, заказ автоматически аннулируется. Менеджер анализирует доходы гостиницы за выбранный период времени (в том числе и по видам номеров), просматривает загруженность номеров различных видов за выбранный период.

2.15. Информационная система «Расписание занятий»

Приложение должно позволять формировать расписание занятий на семестр для введённых дисциплин с количеством пар по каждому виду занятий, групп (подгрупп), преподавателей, лекционных аудиторий и лабораторий (компьютерных или специализированных). Приложение

должно проверять различные факторы: накладки преподавателей, групп (подгрупп), аудиторий (с учётом максимальной вместимости аудиторий и количества студентов в подгруппе, совпадением вида аудитории с необходимым для данной дисциплины и вида занятия).

2.16. Информационная система «Авиакасса»

Приложение должно позволять хранить список городов, между которыми осуществляются авиаперелёты, список маршрутов, рейсов и самолётов, а также формировать необходимые бригады (в зависимости от типа самолёта). Также приложение должно позволять учитывать загрузку самолётов различных типов (пассажирских, грузо-пассажирских, грузовых) с учётом дальности полёта и расхода топлива (учитывая массу топлива, требуемую на данный рейс). Для пассажиров должна быть доступна система продажи и предварительных заказов билетов на рейс (с условием выкупа билета за максимум 2 часа до вылета).

2.17. Информационная система «Агентство недвижимости»

Приложение должно позволять сохранять информацию о различных заявках на продажу объектов недвижимости и о заявках на покупку объектов недвижимости. Для имеющихся заявок система должна подбирать подходящие объекты, исходя из различных характеристик (место расположение, тип объекта недвижимости, площадь и т.д.). В случае если на один объект недвижимости подано несколько заявок на покупку, подтверждённых после непосредственного ознакомления покупателя с объектом недвижимости, система должна организовать электронный аукцион. За каждую заключённую сделку агентство получает некоторый процент прибыли, фиксируемый при подаче заявки на продажу (при подтверждении работником агентства). Приложение должно предоставлять отчёты за требуемый период о количестве заключённых сделок и суммарном доходе от них.

2.18. Информационная система «Автобаза»

Приложение должно хранить список транспортных средств, водителей, маршрутных листов на рейсы, ремонтов транспортных средств. Исходя из этой информации необходимо рассчитывать периоды занятости того или иного транспортного средства (транспортные средства в рейсе или на ремонте не могут быть доступны) или водителя (водитель в рейсе недоступен, и после рейса водитель должен иметь отдых, равный по

длительности половине длительности рейса). При подаче клиентов заявки на перевозку у клиента должна быть возможность сформировать маршрут из имеющихся пунктов. На основе анализа вида и длины дороги между пунктами в маршруте и исходя из массы требуемого груза, расходе топлива подобранного транспортного средства, должна рассчитываться стоимость перевозки.

2.19. Информационная система «Банк»

Приложение должно позволять хранить список клиентов, их счетов и дебитных карт для этих счетов, а также позволять клиентам осуществлять платежи через банковскую систему, перевод средств с одного счёта на другой (с вычетом налогов и комиссии исходя из вида трансфера и суммы). Так, например, отдельно комиссия может рассматриваться на платежи, на перевод средств между физическими и юридическими лицами, к тому же комиссия по разным видам трансферов комиссия может выставляться как в виде процентной ставки от суммы, так и в виде фиксированной суммы. Ставка налога также зависит от вида трансфера и суммы, и также может быть фиксированной или процентной. Система должна представлять отчёты за выбранный период: о наличии средств на счетах клиентов банках, об объёме трансферов по различным видам, об объёмах комиссионных и налоговых сборов. Клиентам предоставляется возможность просмотреть историю всех своих трансферов за выбранный период.

2.20. Информационная система «Бухгалтерия»

Приложение должно позволять вести историю приходов и расходов денежных средств семьи по различным категориям. Рассчитывать дебет и кредит за выбранные периоды, анализировать доходы и расходы по категориям. Исходя из статистики изменения расходов и доходов по выбранным категориями, прогнозировать дебет и кредит на выбранный период в будущем.

ЛИТЕРАТУРА

1. Гамма, Э. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес; [пер. с англ. А. Слинкин]. – СПб.: Питер, 2014. – 368 с.
2. Маклавлин, Б. Объектно-ориентированный анализ и проектирование / [пер. с анг. Е. Матвеева]. – СПб.: Питер, 2013. – 608 с.
3. Фаулер, М. Шаблоны корпоративных приложений / М. Фаулер [и др.]. – М.: Вильямс, 2010. – 544 с.
4. Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер [и др.]. – М.: Вильямс, 2007. – 544 с.
5. Шмуллер, Д. Освой самостоятельно UML за 24 часа / Д. Шмуллер. – М.: Вильямс, 2005. – 405 с.
6. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон. – 2-е издание. – СПб.: ДМК Пресс, 2004. – 432 с.
7. Фаулер, М. UML. Основы / М. Фаулер, К. Скотт. – СПб.: Символ-Плюс, 2002. – 192 с.
8. Скотт, К. UML. Основные концепции / К. Скотт. – М.: Вильямс, 2002. – 138 с.

Учебное издание

КАЗАНЦЕВА Ольга Геннадьевна
НОВЫЙ Вадим Владимирович
ЕРМОЧЕНКО Сергей Александрович

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Методические рекомендации

Технический редактор *Г.В. Разбоева*
Компьютерный дизайн *Т.Е. Сафранкова*

Подписано в печать .2015. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.
Усл. печ. л. 2,49. Уч.-изд. л. 2,20. Тираж экз. Заказ .

Издатель и полиграфическое исполнение – учреждение образования
«Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя,
изготовителя, распространителя печатных изданий
№ 1/255 от 31.03.2014 г.

Отпечатано на ризографе учреждения образования
«Витебский государственный университет имени П.М. Машерова».
210038, г. Витебск, Московский проспект, 33.