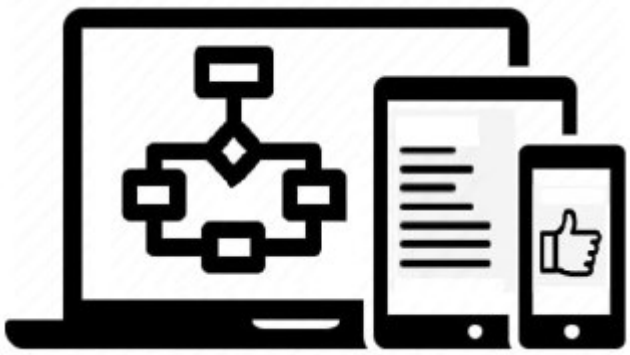
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

АРХИТЕКТУРА ПРОГРАММНЫХ СИСТЕМ

Методические рекомендации к лабораторным работам для студентов направления подготовки 09.03.04 «Программная инженерия» очной формы обучения



Могилев 2023

УДК 004.65
ББК 32.972
А87

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «29» ноября 2022 г., протокол № 5

Составители: канд. техн. наук, доц. К. В. Овсянников;
канд. техн. наук, доц. К. В. Захарченков;
канд. техн. наук, доц. Т. В. Мрочек

Рецензент канд. техн. наук, доц. В. М. Ковальчук

Методические рекомендации содержат описание пяти лабораторных работ по дисциплине «Архитектура программных систем» для студентов направления подготовки 09 03 04 «Программная инженерия» очной формы обучения.

Учебно-методическое издание

АРХИТЕКТУРА ПРОГРАММНЫХ СИСТЕМ

Ответственный за выпуск	В. В. Кутузов
Корректор	И. В. Голубцова
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Современное состояние науки и техники требует от инженерно-технических и научных работников знания средств вычислительной техники и умения обращения с современными программно-техническими комплексами. Эффективное применение компьютеров для решения инженерных и научных задач невозможно без знаний основных методов составления схем алгоритмов, написания действенного программного обеспечения (ПО) на языке программирования, использования пакетов программ инженерной графики и математических систем.

Целью изучения дисциплины является изучение системы инженерных принципов для создания ПО с заданными характеристиками.

Задачи лабораторного цикла работ:

– участие в проектировании компонентов программного продукта – информационной системы (ИС) в объеме, достаточном для их конструирования в рамках поставленного задания;

– создание компонентов ПО (кодирование, отладка, модульное и интеграционное тестирование);

– выполнение измерений и рефакторинг кода в соответствии с планом;

– участие в интеграции компонент программного продукта;

– разработка и оформление эскизной, технической и рабочей проектной документации.

После выполнения каждой лабораторной работы студент оформляет отчет. Отчет по лабораторной работе включает: название и цель работы, краткий порядок действий, ответы на контрольные вопросы.

Полученные при изучении дисциплины знания и навыки могут быть востребованы при курсовом проектировании и в дальнейшем процессе обучения студента в вузе.

1 Проблемы создания сложных программных систем

Цель работы: составить документ описания требований к разрабатываемой информационной системе.

Краткие теоретические сведения

1.1 Установление требований

1.1.1 Результатом этапа установления требований является **документ, описывающий требования**. Большинство организаций вырабатывает документ описания требований в соответствии с заранее определенным шаблоном. Шаблон определяет структуру (содержание) и стиль документа [1, 2, 4, 7, 8, 12].

Основу документа описания требований составляют формулировки (изложение) требований. Требования могут быть сгруппированы в виде *формулировок сервисов* (часто называемых функциональными требованиями) и *формулировок ограничений*. Формулировки сервисов могут быть затем разделены на *требования к функциям* (function requirements) и *требования к данным* (data requirements) (термин «функциональные требования» (functional requirements) в широком и в узком смысле используется как взаимозаменяемый. При использовании в узком смысле он соответствует тому, что называют требованиями к функциям).

Во вводной части документа рассматривается бизнес-контекст проекта, включая цель проекта, участников проекта и основные ограничения. Ближе к заключительной части описываются другие проектные вопросы, включая план-график выполнения проектных работ, бюджет, риски, документацию и т. д.

1.1.2 Шаблоны. Шаблоны для документов описания требований можно найти в учебниках, стандартах ISO, IEEE и т. д., на Web-страницах консалтинговых фирм, программных средствах разработки и т. д. Со временем каждая организация разрабатывает свои собственные стандарты, которые соответствуют принятой в организации практике, корпоративной культуре, типам разрабатываемых систем и т. д. Шаблон документа описания требований определяет структуру документа и содержит подробные указания о содержании каждого из разделов документа. Указания могут включать содержание вопросов, мотивацию, примеры и дополнительные соображения.

На рисунке 1.1 показана типичная структура документа описания требований. Последующие разделы включают объяснение к приведенному оглавлению.

1.1.3 Предварительные замечания к проекту. Часть документа описания требований, содержащая предварительные замечания к проекту, преимущественно дает ориентиры тем руководителям и участникам проекта, ответственным за принятие решений, которые, вероятно, не станут подробно изучать документ целиком. В начале документа необходимо ясно определить цели и рамки проекта, а затем описать деловой контекст системы.

Документ описания требований должен создать прецедент для системы [9]. В частности, необходимо упомянуть обо всех усилиях, приложенных для обос-

нования необходимости системы на этапе планирования системы. Документ должен прояснить вопрос о том, как предлагаемая система может способствовать достижению деловых целей и решению задач организацией.

Документ описания требований
Содержание документа
1 Предварительные замечания к проекту
1.1 Цели и рамки проекта
1.2 Деловой контекст
1.3 Участники проекта
1.4 Идеи в отношении решений
1.5 Обзор документа
2 Системные сервисы
2.1 Рамки системы
2.2 Функциональные требования
2.3 Требования к данным
3 Системные ограничения
3.1 Требования к интерфейсу
3.2 Требования к производительности
3.3 Требования к безопасности
3.4 Эксплуатационные требования
3.5 Политические и юридические требования
3.6 Другие ограничения
4 Проектные вопросы
4.1 Открытые вопросы
4.2 Предварительный план-график
4.3 Предварительный бюджет
Приложения
Глоссарий
Деловые документы и формы
Ссылки

Рисунок 1.1 – Структура документа описания требований

Необходимо обозначить участников проекта системы. Важно, чтобы заказчик выступал не в виде безликого подразделения или офиса – необходимо привести конкретные имена.

Хотя документ описания требований может быть как угодно далек от технических решений, все же важно описать идеи, касающиеся решения на самых ранних этапах жизненного цикла (ЖЦ) разработки.

Документ описания требований должен предоставлять перечень существующих программных пакетов и компонент, которые должны быть в дальнейшем изучены в качестве вариантов возможных решений.

Приобретение готового решения изменяет процесс разработки, однако это не избавляет от необходимости анализа требований и проектирования системы.

Наконец, в заключение раздела предварительных замечаний к проекту документа описания требований приводится обзор оставшейся части документа. Это может подтолкнуть к тому, чтобы изучить остальные части документа, а также способствует лучшему пониманию содержания документа. Обзор также может содержать пояснения в отношении методологии анализа проектирования, выбранной разработчиками.

1.1.4 Системные сервисы. Основная часть документа описания требований посвящена определению системных сервисов. Эта часть может занимать до половины всего объема документа. Это, пожалуй, единственная часть документа, которая может содержать обобщенные модели – модели бизнес-требований.

Рамки системы можно моделировать с помощью диаграммы контекста. В пояснениях к диаграмме контекста должны быть четко определены рамки системы. Без подобного определения проект не может быть застрахован от попыток «растянуть» его рамки.

Функциональные требования можно моделировать с помощью диаграммы бизнес-прецедентов. Однако диаграмма охватывает перечень функциональных требований только в самом общем виде. Все требования необходимо обозначить, классифицировать и определить.

Требования к данным можно моделировать с помощью диаграммы бизнес-классов. Так же, как и в случае функциональных требований, диаграмма бизнес-классов не дает полного определения структур данных для бизнес-процессов. Каждый бизнес-класс требует дальнейших пояснений. Необходимо описать атрибутное наполнение классов и определить идентифицирующие атрибуты классов. В противном случае невозможно правильно представить ассоциации.

1.1.5 Системные ограничения. Системные сервисы определяют, что должна делать система. Системные ограничения определяют, насколько система ограничена при выполнении обслуживания. Системные ограничения связаны со следующими видами требований:

– требования к интерфейсу;
– требования к производительности;
– требования к безопасности;
– эксплуатационные требования;
– политические и юридические требования.

Требования к интерфейсу определяют, как система взаимодействует с пользователями. В документе описания требований определяется только «впечатление и ощущение» от GUI-интерфейса.

Начальное проектирование (закрашивание экрана) GUI-интерфейса ведется во время спецификации требований и позже в ходе системного проектирования.

В зависимости от области приложения требования к производительности могут играть довольно значительную роль в успехе проекта. В узком смысле они задают скорость (время отклика системы), с которой должны выполняться различные задания. В широком смысле, требования к производительности включают другие ограничения – в отношении надежности, готовности, пропускной способности и т. д.

Требования к безопасности описывают пользовательские права доступа к информации, контролируемые системой. Пользователям может быть предоставлен ограниченный доступ к данным или ограниченные права на выполнение определенных операций с данными.

Эксплуатационные требования определяют программно-техническую среду, если она известна на этапе проектирования, в которой должна функционировать система. Эти требования могут оказывать влияние на другие стороны проекта, такие как подготовка пользователей и сопровождение системы.

Политические требования и юридические требования скорее подразумеваются, чем явно формулируются в документе описания требований. Подобная ошибка может обойтись очень дорого. Пока эти требования не выведены явно, программный продукт может быть трудно развернуть по политическим или юридическим причинам.

Возможны и другие виды ограничений. Например, в отношении некоторых систем могут предъявляться повышенные требования к легкости их использования (требования в отношении пригодности к использованию) или легкости их сопровождения (требования в отношении пригодности к сопровождению).

Значение выработки недвусмысленных определений для системных ограничений трудно переоценить. Существует немало примеров проектов, которые провалились из-за упущенных или неверно понятых ограничений. Эта проблема в равной мере относится как к заказчикам, так и к разработчикам. Недобросовестные разработчики могут разыграть «карту системных ограничений», чтобы получить преимущество в своем стремлении уклониться от ответственности.

1.1.6 Проектные вопросы. Заключительная часть документа описания требований обращается к другим проектным вопросам. Один из важных разделов этой части называется «Открытые вопросы». Здесь поднимаются все вопросы, которые могут сказаться на успехе проекта и которые не рассматривались в других разделах документа. Сюда относится ожидаемое возрастание значения некоторых требований, которые в текущий момент выходят за рамки проекта. Сюда можно отнести также любые потенциальные проблемы и отклонения в поведении системы, которые могут начаться в связи с развертыванием системы.

В этой же части необходимо представить предварительный план-график выполнения основных проектных заданий. Сюда же относится предварительное распределение людских и других ресурсов. Для выработки стандартных плановых графиков можно использовать программные средства управления проектами, например, такие как система PERT (program evaluation and review technique – метод оценки и пересмотра планов) или карты Ганта.

Прямым результатом составления план-графика может быть разработка предварительного бюджета. Стоимость проекта может быть выражена скорее в виде диапазона значений затрат, а не конкретного значения. При наличии надлежащим образом документированных требований для оценки затрат можно использовать один из подходящих методов.

1.1.7 Приложения. Приложения содержат остальную, полезную для понимания требований, информацию. Основным здесь является глоссарий, который определяет термины, сокращения и аббревиатуры, используемые в документе

описания требований. Неверное истолкование терминологии таит в себе большую опасность для проекта.

Одна из особенностей, которую часто упускают из виду при составлении документа описания требований, состоит в том, что в проблемной области, определяемой документом, можно довольно неплохо разобраться с помощью изучения документов и форм, используемых в процессах делопроизводства. При возможности следует включать в документ заполненные формы – «пустые» формы не дают такого же уровня понимания бизнес-процессов.

Раздел ссылок содержит перечень документов, которые упоминаются или используются при подготовке документа описания требований. К ним могут относиться книги и другие опубликованные источники информации, но – что, пожалуй, даже более важно – необходимо также упомянуть протоколы совещаний, служебные записки и внутренние документы.

1.2 Пример документа описания требований

Документ описания требований ИС «Домашняя Бухгалтерия» [5, 10, 12].

1 Предварительные замечания к проекту.

1.1 Цели и рамки проекта.

Целью данного проекта является разработка ИС для ведения и оптимизации семейного бюджета. ИС «Домашняя Бухгалтерия» должна быть проста в использовании и не требовать от пользователя знаний бухгалтерского учета.

1.2 Деловой контекст.

Многие семьи в наше время планируют семейный бюджет. Ведение семейного бюджета при помощи подручных средств – карандаш, бумага – не всегда удобно и всегда трудоемко. Использование для этих целей компьютерных программ для ведения бухгалтерии не оправдано с точки зрения сложности их освоения и избыточного функционала для ведения домашней бухгалтерии. В связи с этим возникает необходимость создания специализированной программы ведения домашней бухгалтерии.

1.3 Участники проекта.

Заказчик – Иванов Иван Иванович (i.ivanov@mail.ru).

Разработчик – Петров Петр Петрович (petrov@mysoft.com).

1.4 Идеи в отношении решений.

Программа должна быть реализована в виде настольного приложения для операционных систем семейств MS Windows.

1.5 Обзор документа.

В разделе «Системные сервисы» описывается, что должна делать система.

В разделе «Системные ограничения» определяется, насколько система ограничена при выполнении обслуживания. В разделе «Проектные вопросы» освещаются прочие проектные вопросы.

2 Системные сервисы.

2.1 Рамки системы.

Рамки системы можно моделировать с помощью диаграммы контекста (рисунок 1.2).

Рисунок 1.2 – Контекстная диаграмма ИС «Домашняя Бухгалтерия»

ИС «Домашняя Бухгалтерия» получает данные о доходах и расходах от внешней сущности «Домохозяин». Для передачи этих данных сущности «Домохозяин» должен авторизоваться. В своей работе сущность «Домашняя Бухгалтерия» использует информацию о ценах на товары и тарифах и курсах валют, получаемую от внешних сущностей «База тарифов и цен на товары» и «База курса валют». Результаты своей работы ИС «Домашняя Бухгалтерия» может отображать как внешней сущности «Домохозяин», так и генерировать в виде отчетов формата MS Excel для внешней сущности MS Excel.

2.2 Функциональные требования.

ИС должна обеспечивать следующие функциональные возможности:

– учет расходов;
– учет доходов;
– учет денег, отданных и взятых в долг;
– погашение долгов частями;
– проценты по долгам;
– контроль возврата долгов;
– система напоминания по долгам;
– составление бюджета расходов и доходов;
– планирование расходов;
– планирование доходов;
– система счетов;
– возможность использовать до пяти валют включительно;
– получение курсов валют из интернет;
– обмен валют;
– импорт данных из файлов Microsoft Excel;
– поиск по базе данных;
– фильтры и быстрый поиск по базе данных;
– экспорт данных в Excel, XML, текстовый файл;

– перенос данных;
– резервное копирование;
– печать данных;
– построение отчетов и диаграмм;
– настройка пользовательского интерфейса.

2.3 Требования к данным.

ИС должна хранить свои данные в специализированных XML-файлах.

3 Системные ограничения.

3.1 Требования к интерфейсу ИС должна иметь стандартный интерфейс приложений, разработанных для ОС MS Windows.

3.2 Требования к производительности.

Особых требований к производительности ИС нет.

3.3 Требования к безопасности.

С программой могут работать несколько человек, входя в программу под своими именами. Для обеспечения конфиденциальности каждое имя можно защитить паролем. Добавление, изменение и удаление пользователей осуществляется в администраторе пользователей.

3.4 Эксплуатационные требования.

ИС должна функционировать на ОС Windows 10 и выше. Минимальные аппаратные требования определяются минимальными аппаратными требованиями к вышеперечисленным ОС.

3.5 Политические и юридические требования.

Нет.

3.6 Другие ограничения.

Нет.

Проектные вопросы.

4.1 Открытые вопросы.

Нет.

Предварительный план-график.

01.09.2023 – 01.10.2023 – Анализ и установление требований к ИС.
01.10.2023 – 01.11.2023 – Спецификация требований к ИС.
01.11.2023 – 01.12.2023 – Кодирование ИС.
01.12.2023 – 31.12.2023 – Тестовая эксплуатация ИС.
11.01.2024 – 14.12.2024 – Ввод в эксплуатацию.

Предварительный бюджет.

Пять тысяч белорусских рублей.

5 Приложения.

Глоссарий.

ИС – информационная система.
ОС – операционная система.

Деловые документы и формы.

Нет.

Ссылки.

Нет.

Задание

Предложить для разработки информационную систему. ИС должна представлять собой программный комплекс, наделенный функциональностью, автоматизирующей конкретную деятельность в рамках предметной области, для которой разрабатывается система. Примером таких систем могут служить автоматизированные системы управления (АСУ), интернет-магазины, сервисы.

Необходимо составить документ описания требований к разрабатываемой ИС согласно шаблону.

Содержание отчета: титульный лист; тема и цель работы; текст индивидуального задания; выполнение индивидуального задания.

Контрольные вопросы

1 Определить состав и структуру требований к программным системам.
2 Охарактеризовать состав и структуру системных сервисов.
3 Как определить рамки программной системы?
4 Как определить функциональные требования к программной системе?
5 Как определить требования к данным?
6 Описать состав и структуру системных ограничений.
7 Описать требования к интерфейсу, производительности и безопасности программной системы. Как определить эксплуатационные, политические и юридические требования к программной системе?

2 Архитектурные структуры и представления

Цель работы: изучить существующие известные типы архитектур и определить архитектуру разрабатываемой ИС.

Краткие теоретические сведения

2.1 Понятие и типы архитектуры ИС

Применительно к техническим системам **архитектуру** можно рассматривать, как абстрактную модель, в которой отсутствуют подробности реализации. **Архитектуру ИС** можно описать как «…концепцию, определяющую модель, структуру, выполняемые функции и взаимосвязь компонентов» ИС [10]. Для проектируемой ИС процедура выбора архитектуры сводится к определению стоимости владения ИС. Стоимость владения ИС складывается из плановых затрат (стоимость технического обслуживания, модернизации, зарплата обслуживающего персонала и т. д.) и стоимости рисков. Совокупная стоимость рисков определяется из стоимости всех типов рисков, их вероятностей и матрицы соответствия между ними (матрица определяется выбранной архитектурой ИС).

Далее перечислены наиболее важные типы рисков:

– проектные риски (риски при проектировании и разработке ИС);

– риски разработки (ошибки, недостаточная оптимизация);

– технические риски (простои, отказы, утрата данных);

– бизнес-риски (возникают из-за технических рисков и связаны с эксплуатацией ИС);

– неопределенности (связаны с вариативностью бизнес-процессов и определяются необходимостью внесения изменений в систему);

– операционные (подразумевают невыполнение набора операций, могут возникать из-за технических рисков и являться инициаторами бизнес-рисков).

Концепция архитектуры ИС должна формироваться на этапе технико-экономического обоснования и обеспечивать минимальную стоимость владения.

При определении архитектуры ИС необходимо ответить на ряд вопросов.

Что делает ИС?

На какие составные части разделена система?

Каким образом взаимодействуют эти составные части? Как и где эти части размещены?

Таким образом, архитектура ИС является «…моделью, определяющей стоимость владения через имеющуюся в данной системе инфраструктуру» [10].

При рассмотрении архитектуры крупных организаций используется понятие «корпоративная архитектура», которое можно представить в виде совокупности пяти *типов архитектур* (рисунок 2.1).

Бизнес-архитектура	Уровень 5	Здесь определяются стратегии ведения бизнеса, способы управления, принципы организации и ключевые процессы
ИТ-архитектура	Уровень 4	Если некоторая функция требуется сразу в нескольких приложениях, то ее следует перенести на уровень ИТ-архитектуры, тем самым повысив интеграцию системы и снизив сложность архитектуры приложений. Здесь формируется базовый набор сервисов, которые используются на уровне программной архитектуры и на уровне архитектуры данных для достижения бизнес-целей
Архитектура данных (Data architecture)	Уровень 3	Сюда входят физические и логические хранилища данных, средства управления данными. Здесь описываются логические и физические модели данных, определяются правила целостности и ограничения для данных. Для работы со знаниями может быть выделен особый уровень – архитектура знаний (Knowledge architecture)
Программная архитектура	Уровень 2	Необходим для описания приложений, входящих в состав ИС. Здесь описывают программные интерфейсы, компоненты и поведение (Software architecture)
Техническая архитектура (Hardware architecture)	Уровень 1	Описывает все аппаратные средства, использующиеся при выполнении заявленного набора функций ИС, а также включает средства обеспечения сетевого взаимодействия и надежности (указываются периферийные устройства, сетевые коммутаторы и маршрутизаторы, жесткие диски, оперативная память, процессоры, соединительные кабели, источники бесперебойного питания и т. п.)

Рисунок 2.1 – Модель корпоративной архитектуры

2.2 Микроархитектура и макроархитектура

Микроархитектура описывает внутреннее устройство конкретного компонента или подсистемы. Микроархитектуру можно отнести к уровням программной архитектуры и архитектуры данных [4, 5, 10, 12].

Макроархитектура описывает устройство всей ИС, как совокупности ее компонентов или подсистем. Макроархитектуру можно отнести к уровню ИТ-архитектуры.

При построении сложных систем без применения какого-либо архитектурного подхода их создание, обслуживание и модификация могут стать нерентабельными для бизнеса. Для решения данной задачи можно использовать методы абстракции, декомпозиции, инкапсуляции.

Так, при разработке программной системы, входящей в состав инфраструктуры крупной организации, она представляется в виде множества модулей, каждый из которых выполняет определенную функцию, а все вместе они выполняют функции самой системы. Организация каждого модуля будет являться микроархитектурой, а способы взаимодействия между этими модулями в рамках системы – макроархитектурой.

Уменьшение сложности реализации системы будет происходить за счет разбиения сложных задач на несколько более простых. В результате это может привести к появлению большого набора простых задач, на основании которого можно будет реализовать любую, даже очень сложную систему. В этом заключается принцип объектно-ориентированного программирования – создание конкретных классов объектов для решения конкретных задач.

Не всегда сформированная система классов сможет упростить процессы управления программной системой. Если подсистема выполняет слишком большой спектр действий, необходимых для различных процессов в организации, выход ее из строя нарушает функционирование сразу нескольких из них.

Известны два принципа, позволяющие оценить взаимное влияние компонентов системы друг на друга: **Low Coupling** (слабая связанность) и **High Cohesion** (сильное зацепление).

Принцип **Low Coupling** способствует распределению функций между компонентами системы таким образом, чтобы степень связанности между ними оставалась низкой. **Степень связанности** (coupling) – это мера взаимозависимости подсистем. Принцип Low Coupling связан с одним из основных принципов системного подхода, который требует минимизации информационных потоков между подсистемами.

Подсистема с **низкой степенью связанности** (или слабым связыванием) характеризуется следующими свойствами:

– малое число зависимостей между подсистемами;

– слабая зависимость одной подсистемы от изменений в другой;

– высокая степень повторного использования подсистем.

В свою очередь, принцип **High Cohesion** задает свойство сильного зацепления внутри подсистемы. В результате подсистемы получаются сфокусированными, управляемыми и понятными. **Зацепление** (функциональное зацепление) –

это мера связанности и сфокусированности функций подсистемы. Подсистема обладает высокой степенью зацепления, если ее функции тесно связаны между собой, и она не выполняет больших объемов работы.

Подсистема с **низкой степенью зацепления** выполняет множество различных функций, никак не связанных между собой. Такие подсистемы создавать нежелательно, поскольку они приводят к возникновению следующих проблем:

– трудность понимания.

– сложность при повторном использовании.

– сложность поддержки.

– ненадежность, постоянная подверженность изменениям.

Подсистемы с низкой степенью зацепления не имеют четкого функционального назначения и выполняют слишком разноплановые функции, которые можно легко распределить между другими подсистемами.

Связанность (coupling) и зацепление (cohesion) являются общесистемными характеристиками и могут применяться при проектировании любых систем. Связанность является характеристикой системы целиком, а зацепление характеризует отдельно взятую подсистему.

2.3 Подходы к проектированию ИС

Процесс проектирования ИС тесным образом связан с ее архитектурным описанием [5, 10]. Выделяют пять подходов к проектированию (таблица 2.1).

Таблица 2.1 – Подходы к проектированию ИС

Название	Характеристика
Календарный подход	Предполагает составление графика предстоящих работ с их поэтапным выполнением. Ключевые решения принимаются на основе локальных цели и задач каждого этапа разработки. Практически не отводится время на разработку документации, формирование архитектуры и процессов по внесению различных изменений. Из-за этого в долгосрочной перспективе возрастает стоимость владения разработанной ИС. Такой подход считается морально устаревшим, однако в некоторых компаниях до сих пор применяется
Подход, за основу которого взят процесс управления требованиями	Бóльшая часть времени процесса разработки уходит на функциональные характеристики системы, а нефункциональные, такие как масштабируемость, например, практически не рассматриваются. Все решения в ходе проекта формируются исходя из локальных целей по реализации конкретного функционала. Такой подход может быть эффективен, если требования к разрабатываемой системе определены заранее и не меняются при проектировании. Недостатки: несоответствие стандарту качества ISO 9126; нестабильность разрабатываемых архитектур, т. к. каждая реализуемая функция связывается с одним или несколькими компонентами. В связи с этим трудоемкость при добавлении к подобным системам дополнительных функций возрастает. Применение этого подхода в долгосрочной перспективе является нерациональным, однако позволит успешно управлять требованиями в рамках требуемой функциональности

Окончание таблицы 2.1

Название	Характеристика
Подход, основанный на процессе разработки документации	Неоправданно большое количество времени тратится на формирование пакета документов, которые часто не используются ни заказчиком, ни пользователем. Кроме того, из-за нехватки времени страдает качество самой разрабатываемой системы. Данный подход используется в правительственных организациях и крупных компаниях
Подход, в основе которого лежит система управления качеством	Включает в себя большое количество разнообразных мер для отслеживания параметров, наиболее значимых для функционирования системы. Выбранные параметры наблюдаются на всех стадиях разработки системы, иногда в ущерб другим. Основной недостаток: иногда набор такого рода параметров может быть составлен неправильно, и оптимизация системы будет проведена ошибочно. Кроме этого, при появлении новых требований весьма трудно изменять функциональность, а значит, дальнейшее развитие системы, в том числе и из-за архитектурных просчетов, может быть невозможно. Такой подход считается консервативным, а его применение целесообразно при необходимости создать систему с экстремальными характеристиками
Архитектурный подход	Считается наиболее зрелым. Его ключевым аспектом является создание фреймворка, т. е. каркаса, адаптация которого под нужды конкретной системы легко осуществима. В соответствии с этим задача проектирования разбивается на две: разработка многократно используемого каркаса и создание системы на его основе. Данные подзадачи могут решаться разными группами специалистов. Каркасы предоставляют возможность довольно быстро изменять функциональность системы за счет итеративности процесса проектирования. Архитектурный подход призван ликвидировать недостатки, возникающие в процессе проектирования, основанном на управлении требованиями

Из всех рассмотренных подходов к проектированию сложно однозначно определить лучший, т. к. каждая конкретная проектная группа и каждая проектируемая ИС имеют свои особенности, на основании которых следует выбирать тот подход, применение которого позволит решить поставленные задачи в установленные сроки и в рамках выделенного бюджета.

2.4 Значение ПО в ИС. Характеристики качества ПО

Пользователи современных ИС почти всегда взаимодействуют с ними с помощью специальных программных модулей, от показателей качества которых зависит уровень качества всей ИС целиком. Существует огромное количество стандартов для создания правильной и надежной архитектуры, а также для разработки и интеграции программных систем.

Качеством ПО в соответствии со стандартом ISO 9126 можно считать совокупность его характеристик, относящихся к возможности удовлетворять высказанные или подразумеваемые потребности всех заинтересованных лиц [5, 10].

Можно выделить три аспекта качества:

1) внутреннее качество (характеристики самого ПО);

2) внешнее качество (поведенческие характеристики ПО);

3) контекстное качество (ощущения пользователей при различных контекстах использования).

Стандарт ISO 9126 выделяет шесть характеристик, описывающих **внутреннее** и **внешнее** качество ПО (таблица 2.2).

Таблица 2.2 – Характеристики внутреннего и внешнего качества ПО

Характеристика	Подхарактеристика
1 Функциональность: подразумевает способность ПО решать задачи в определенных условиях	1.1 Функциональная пригодность (suitability) – способность решать нужный набор задач
	1.2 Точность (accuracy) – способность получать требуемые результаты
	1.3 Способность к взаимодействию (interoperability) – способность взаимодействия с требуемым набором иных систем
	1.4 Защищенность (security) – способность предотвращать неавторизованный доступ к данным и программам
	1.5 Соответствие стандартам и правилам (compliance) – соответствие ПО различным регламентирующим нормам
2 Надежность (reliability) характеризуется способностью ПО удерживать функциональность в заданных рамках при определенных условиях	2.1 Зрелость (maturity) – величина, обратная частоте отказов ПО
	2.2 Устойчивость к отказам (fault tolerance) – способность удерживать определенный уровень работоспособности при различных отказах и нарушениях правил взаимодействия с окружением
	2.3 Способность к восстановлению (recoverability) – способность восстанавливать требуемый уровень работоспособности после отказа
	2.4 Соответствие стандартам надежности (reliability compliance)
3 Производительность: (efficiency) определяется способностью ПО при определенных условиях гарантировать требуемую работоспособность по отношению к выделяемым для этого ресурсам	3.1 Временная эффективность (time behavior) – способность ПО получать требуемые результаты и обеспечивать передачу необходимого объема данных за определенное время
	3.2 Эффективность использования ресурсов (resource utilization) – способность программного обеспечение решать требуемые задачи и использованием заданных объемов определенных видов ресурсов
	3.3 Соответствие стандартам производительности (efficiency compliance)
4 Удобство использования (usability): характеризуется привлекательностью для пользователей, удобством в обучении и использовании ПО	4.1 Понятность (understandability) – величина обратная усилиям, затраченным пользователями, по осознанию применимости ПО для решения требуемых задач
	4.2 Удобство работы (operability) – величина обратная усилиям, затраченным пользователями, для решения требуемых задач при помощи ПО
	4.3 Удобство обучения (learnability) – величина обратная усилиям, затраченным пользователями, на процесс обучения работе с программным обеспечением
	4.4 Привлекательность (attractiveness) – способность ПО быть привлекательным для пользователей
	4.5 Соответствие стандартам удобства использования (usability compliance)

Окончание таблицы 2.2

Характеристика	Подхарактеристика
5 Удобство сопровождения (maintainability): характеризуется удобством сопровождения ПО	5.1 Анализируемость (analyzability) характеризуется удобством проведения анализа ошибок, дефектов, недостатков, необходимостей внесения изменений и их возможных последствий
	5.2 Удобство внесения изменений (changeability) – величина обратная трудозатратам на выполнение требуемых изменений
	5.3 Стабильность (stability) – величина, обратная риску появления непредусмотренных последствий при внесении требуемых изменений
	5.4 Удобство проверки (testability) – величина, обратная требуемым трудозатратам на тестирование и другие виды проверок достижения заданных результатов при внесении изменений
	5.5 Соответствие стандартам удобства сопровождения (maintainability compliance)
6 Переносимость (portability): характеризуется способностью ПО сохранять работоспособность при изменении организационных, аппаратных и программных аспектов окружения	6.1 Адаптируемость (adaptability) – способность ПО без совершения непредусмотренных действий приспосабливаться к изменениям окружения
	6.2 Удобство установки (installability) – способность ПО устанавливаться в заранее определенное окружение
	6.3 Способность к сосуществованию (coexistence) – способность ПО функционировать в общем окружении с другими программами, разделяя с ними ресурсы
	6.4 Удобство замены (replaceability) – возможность применения ПО вместо уже используемого для решения тех же задач, в том же окружении
	6.5 Соответствие стандартам переносимости (portability compliance)

Контекстное качество описывают с помощью следующих характеристик:

– эффективность (effectiveness) – способность ПО решать пользовательские задачи с заданной точностью и в заданном контексте;

– продуктивность (productivity) – способность ПО получать требуемые результаты при использовании заранее определенного количества ресурсов;

– безопасность (safety) – способность ПО поддерживать требуемый низкий уровень риска нанесения ущерба людям, бизнесу и окружающей среде;

– удовлетворенность пользователей (satisfaction) – способность ПО при применении в определенном контексте приносить удовлетворение пользователям.

Руководствуясь рассмотренными показателями, можно значительным образом увеличить качество программных модулей, а значит, и всей ИС в целом.

2.5 Функциональные компоненты ИС

Учитывая принцип декомпозиции, принято проектировать ИС с разделением функционального назначения их компонентов, т. е. создавать многоуровневое представление. Принято выделить три основные функциональные группы, предназначенные для решения различных по смыслу задач [4, 5, 9, 10]:

1) взаимодействие с пользователями;

2) бизнес-логика;

3) управление ресурсами.

Подобный функционал реализуется путем создания соответствующей программной системы, также имеющей многоуровневое представление компонентов (таблица 2.3).

Таблица 2.3 – Компоненты программной системы

Наименование компонента	Назначение компонента
Компонент представления	Обеспечивает взаимодействие пользователей с программой, т. е. обрабатывает нажатия клавиш, движения различных контроллеров, осуществляет вывод информации – предоставляет пользовательский интерфейс
Прикладной компонент	Представляет собой набор правил и алгоритмов реализации функций системы, реакций на действия пользователей или внутренние события
Компонент управления ресурсами	Отвечает за модификацию, хранение, выборку и удаление данных, связанных с решаемой прикладной задачей

2.6 Платформенные архитектуры ИС

Одним из важнейших этапов проектирования архитектуры ИС является распределение этих функциональных компонентов по выбранной платформенной архитектуре. Можно выделить три направления развития платформенных архитектур: **автономные**, **централизованные** и **распределенные** [4, 5, 10].

Автономная архитектура предполагает наличие всех функциональных компонентов системы на одном физическом устройстве (например, компьютере) и не имеет связей с внешней средой. Примеры подобных систем – текстовые редакторы, системные утилиты и простые корпоративные приложения. При построении корпоративной ИС не должны формироваться несвязанные узлы или модули. Однако их появление может быть обусловлено определенными требованиями к надежности или безопасности.

Централизованная архитектура предусматривает выполнение всех задач на специально отведенном вычислительном узле достаточной мощности (мейнфрейм (mainframe)), содержащем прикладной компонент и компонент управления ресурсами. Пользователь работает за терминальной станцией (терминалом), которая содержит компонент представления, выступает только в виде устройства ввода-вывода и не имеет иных функциональных возможностей. Достоинства этой архитектуры: отсутствие необходимости администрирования рабочих мест; легкость обслуживания и эксплуатации системы, т. к. все ресурсы находятся в одном месте. Недостатки: функционирование всей системы полностью зависит от мейнфрейма; все ресурсы и программные средства являются коллективными и не могут изменяться под нужды конкретных пользователей. Для преодоления последнего недостатка в современных ИС применяются технологии виртуализации, позволяющие выделить каждому пользователю необходимое количество

ресурсов и установить требуемое ПО. При этом можно создать практически любую архитектуру, используя только ресурсы мейнфрейма.

Для **распределенной** архитектуры функциональные компоненты ИС распределяются по имеющимся узлам в зависимости от поставленных целей и задач. Выделяют пять основных характеристик архитектуры распределенных систем:

1) совместное использование ресурсов (как аппаратных, так и программных);

2) открытость – возможность увеличения типов и количества ресурсов;

3) параллельность – возможность выполнения нескольких процессов на различных узлах системы (при этом они могут взаимодействовать);

4) масштабируемость – возможность добавлять новые свойства и методы;

5) отказоустойчивость – способность системы поддерживать частичную функциональность за счет возможности дублирования информации, аппаратной и программной составляющей.

Недостатки распределенных систем: структурная сложность; сложность обеспечения достаточного уровня безопасности; большое количество усилий для управления системой; непредсказуемая реакция на изменения. Указанные недостатки связаны в первую очередь со сложной структурой, разноплановым оборудованием и сложной системой распределения прав доступа.

Существуют следующие **виды распределенных архитектур**: архитектура «файл-сервер»; архитектура «клиент-сервер» (двухзвенная и многозвенная); архитектура Web-приложений (Web-сервисов) [1, 3–5, 9, 10].

Архитектура Web-приложений подразумевает предоставление некоторого сервиса, доступного в сети Internet, через специальное приложение. Основой для предоставления таких услуг служат открытые стандарты и протоколы SOAP, UDDI и WSDL [1, 3–5]. Выделяют три технологии, которые можно использовать для построения распределенной архитектуры Web-сервиса: EJB (Enterprise JavaBeans), DCOM (Distributed Component Object Model) и CORBA (The Common Object Request Broker Architecture).

При грамотном подходе к построению архитектуры ИС может потребоваться использование сразу нескольких из рассмотренных технологий. Каждая из них будет реализовывать определенный функционал, который может потребовать также нескольких типов платформенных архитектур. В одной системе могут работать несколько файл-серверов, несколько серверов приложений и несколько серверов баз данных. Таким образом можно распределять нагрузку в системе или группировать наборы сервисов по выполняемым функциям.

Задание

Составить для проектируемой ИС схему архитектуры с обоснованием всех принимаемых решений.

Содержание отчета: титульный лист; тема и цель работы; текст индивидуального задания; выполнение индивидуального задания.

Контрольные вопросы

1 Описать техническую архитектуру программной системы.
2 Описать программную архитектуру ИС.
3 Описать архитектуру данных ИС.
4 Описать архитектуру бизнес-процессов программной системы.
5 Описать функциональность программной системы.
6 Определить надёжность, производительность, удобство использования, удобство сопровождения, переносимость программной системы.
7 Описать микроархитектуру и макроархитектуру программной системы.
8 Применить архитектурный, календарный подходы к проектированию ИС.
9 Применить подход к разработке программных систем, за основу которого взят процесс управления требованиями.
10 Применить подходы к разработке программных систем, основанные на процессе разработки документации и на системе управления качеством.

3 Проектирование программных систем

Цель работы: спроектировать компоненты разрабатываемой ИС.

Краткие теоретические сведения

Все известные методологии проектирования программных систем можно разделить на три большие группы [1, 3–5, 9, 10]:

1) структурные методологии, ориентированные на **первоочередное проектирование функций системы** (процедурно-ориентированные);

2) структурные методологии, ориентированные на **первоочередное проектирование данных системы** (ориентированные на данные);

3) **объектно-ориентированные методологии**.

Методология Йордона является примером структурного подхода, ориентированного на **первоочередное проектирование функций**. По некоторым оценкам эта методология является наиболее часто используемой (36 % проектов). В соответствии с этой методологией весь проект разделяется на четыре фазы (таблица 3.1).

Разработаны модификации методологии Йордона вместе с другими авторами – методологии Йордона – ДеМарко, Йордона – Константайна, Коада – Йордона.

Методология Гейна и **Сарсона** очень близка к методологии Йордона (она применяется в 20 % случаев). Отличием этой методологии является наличие этапа моделирования данных, определяющего содержимое хранилищ данных в диаграммах потоков данных в третьей нормальной форме и включающего:

– построение списка элементов данных, располагающихся в каждом хранилище данных;

– анализ отношений между данными и построение соответствующей диаграммы связей между элементами данных;

– представление всей информации по модели в виде связанных нормализованных таблиц.

Таблица 3.1 – Этапы проектирования в методологии Йордона

Фаза	Результат	Действие в данной фазе
1	2	3
Анализ	Модель среды	Анализ поведения системы: определение назначения системы, построение начальной контекстной диаграммы потоков данных (DFD) формирование матрицы списка событий (ELM – Event List Matrix) построение контекстных диаграмм)
		Анализ данных (определение состава потоков данных и построение диаграмм структур данных (DSD – Data Structure Diagram)
		Конструирование глобальной модели данных в виде ER-диаграммы)
		Учет взаимосвязей между различными типами диаграмм: ELM-DFD (события – входные потоки, реакции – выходные потоки) DFD-DSD (потоки данных – структуры данных верхнего уровня) DFD-ERD (накопители данных – ER-диаграммы) DSD-ERD (структуры данных нижнего уровня – атрибуты сущностей)
Проектирование архитектуры	Предметная модель	Детальное описание функционирования системы. Построение модели процессов (диаграммы архитектуры системы (SAD – System Architecture Diagram) и мини-спецификации на структурированном языке)
		Дальнейший анализ используемых данных и построение логической модели данных для последующего проектирования базы данных. Построение модели данных (ERD и подсхемы ERD)
		Определение структуры пользовательского интерфейса, спецификация форм и порядка их появления. Построение модели пользовательского интерфейса (классификации процессов на интерактивные и неинтерактивные функции), а также диаграммы последовательности форм (FSD – Form Sequence Diagram), показывающей набор и порядок вызовов экранных форм в приложении
Детальное проектирование	Модульная модель – реальная модель проектируемой прикладной системы	Уточнение модели базы данных для последующей генерации SQL-кода, определяющего структуру целевой БД. Построение модели данных (определение в ERD всех нужных параметров для приложений)
		Уточнение структуры пользовательского интерфейса и построение структурных схем, отражающих логику работы пользовательского интерфейса и модель бизнес-логики (SCD – Structure Charts Diagram), и привязка их к формам. Построение модели процессов (структурные схемы интерактивных и неинтерактивных функций), а также модели пользовательского интерфейса (диаграммы последовательности форм FSD), показывающей, какие формы появляются в приложении и в каком порядке, взаимосвязь между каждой формой и определенной структурной схемой, между каждой формой и одной или более сущностью в ERD

Окончание таблицы 3.1

1	2	3
Реализация	Модель реализации системы	Генерация SQL-кода структуры целевой БД (таблицы, индексы, ограничения целостности)
		Уточнение структурных схем (SCD) и диаграмм последовательности форм (FSD) с последующей генерацией кода приложений
		На основе анализа потоков данных и взаимодействия процессов с хранилищами данных осуществляется окончательное выделение подсистем. При выделении подсистем руководствуются принципами функциональной связанности и минимизации информационной зависимости. Необходимо учитывать, что на основании таких элементов подсистемы, как процессы и данные, на этапе разработки должно быть создано приложение, способное функционировать самостоятельно

Методология SSADM (Structured Systems Analysis and Design Method) ориентирована на **первоочередное проектирование функций**. Ее достоинством является наличие взаимосогласованных методик, регламентирующих начальные этапы разработки системы, основным из которых является этап итеративного определения требований. SSADM не распространяется на этапы, связанные с реализацией, внедрением и сопровождением системы, отсылая разработчика к другим методологиям. В SSADM применяется нисходящий подход к построению интегрированных **функциональных** моделей (DFD с мини спецификациями на структурированном естественном языке), **информационных** моделей (в нотации LDS (Logical Data Structure), являющейся диалектом ER-модели) и **событийных** моделей (диаграммы истории жизни сущностей ELN (Entity Life History), поддерживающие индикаторы состояний, события с привязанными к ним действиями, возможность задавать последовательные, параллельные и итеративные конструкции, а также конструкции выбора). Отличием SSADM является четкое выделение и поддержка **нефункциональных требований** (например, среднего времени наработки на отказ; времени отклика; ограничения доступа; требования безопасности и т. п.). Согласно SSADM определение системных требований включает шесть основных этапов (таблица 3.2).

Структурное проектирование Джексона является классическим примером методологии, ориентированной на **первоочередное проектирование данных** [1]. Его базовая процедура проектирования предназначена для «простых» программ («сложная» программа разбивается на «простые» традиционными методами) и включает следующие четыре этапа (таблица 3.3).

Методология DSSD (Data-Structured Systems Development), предложенная Ж.-Д. Варнье и К. Орром, является примером подхода, **ориентированного на данные** [1]. Методология DSSD использует теорию множеств для описания проекта ПО и ориентирована на разработку систем со структурными данными. Так же, как и в математике, множество определяется перечислением его элементов. Подобно методологии Джексона, структура программы строится на базе структур данных.

Таблица 3.2 – Этапы проектирования в методологии SSADM

Этап	Результат	Действие этапа
Оценка реализуемости	Инициализация работ по созданию системы	Анализ первичных бизнес-требований (определение целевого назначения будущей системы, ее основных пользователей и т. п.)
		Предварительная экономическая оценка проекта
		Построение план-графика выполнения основных работ
		Подготовка документов по оценке возможности создания системы
Предпроектное обследование и моделирование требований	Функционально полная модель требований заказчика к будущей системе	Определение границ будущей системы
		Выявление основных требований, оценка важности этих требований для будущего пользователя, оценка необходимых для реализации каждого требования ресурсов
		Выявление процессов обработки информации
		Выявление обрабатываемых данных
		Построение информационно-логической модели требований
		Обобщение результатов и подготовка отчетов
Выбор варианта автоматизации	Вариант автоматизации	На основании результатов (оценок) предшествующего этапа предлагаются несколько (от 3 до 6) вариантов автоматизации, из которых на основании выявленных ограничений совместно с заказчиком выбирается окончательный вариант
Разработка логического проекта	Прототип будущей системы	Пересмотр выявленных требований с учетом выбранного варианта автоматизации с фиксацией неотраженных данным вариантом требований. Требования детализируются и уточняются, выявляются противоречия между ними, создается и оценивается прототип будущей системы. После завершения данного этапа SSADM запрещает добавление новых функциональных требований, допускается лишь их корректировка и уточнение
Выбор варианта реализации	Вариант реализации	Проработка нескольких вариантов реализации, касающихся технической и программной среды, их оценка и совместный с заказчиком выбор приемлемого варианта
Физическое проектирование	Физическая информационная модель и документация	Разработка физической информационной модели
		Разработка спецификаций к программным компонентам
		Оптимизация информационной модели
		Уточнение спецификаций к программным компонентам
		Оформление документации

Однако в методологии Джексона необходимо объединять все входные и выходные структуры данных для продуцирования структуры программы, а в DSSD входные данные и структура программы продуцируются из выходных структур. Таким образом, главная аксиома DSSD утверждает, что выходные структуры данных полно и точно определяют входные структуры, которые, в свою очередь, определяют и логику их обработки. Основные этапы методологии DSSD изображены на рисунке 3.1 с помощью диаграммы Варнье – Орра.

Таблица 3.3 – Этапы методологии структурного проектирования Джексона

Этап	Действие этапа
Проектирование данных	Построение системной сетевой диаграммы, демонстрирующей все хранилища, источники и стоки данных в программе
	Представление каждой входной и выходной структуры данных древовидной структурной диаграммой
Проектирование программ	Формирование структуры программы комбинированием структур данных
	Идентификация всех связей между компонентами структур данных
	Верификация полученной структуры программы
Проектирование операций	Построение списка операций, необходимых для продуцирования выходных структур данных из входных
	Назначение операций компонентам структуры программы
Проектирование кода	Трансляция построенной модели программы в код с добавлением ряда логических условий для управления выполнением циклов и выбором данных

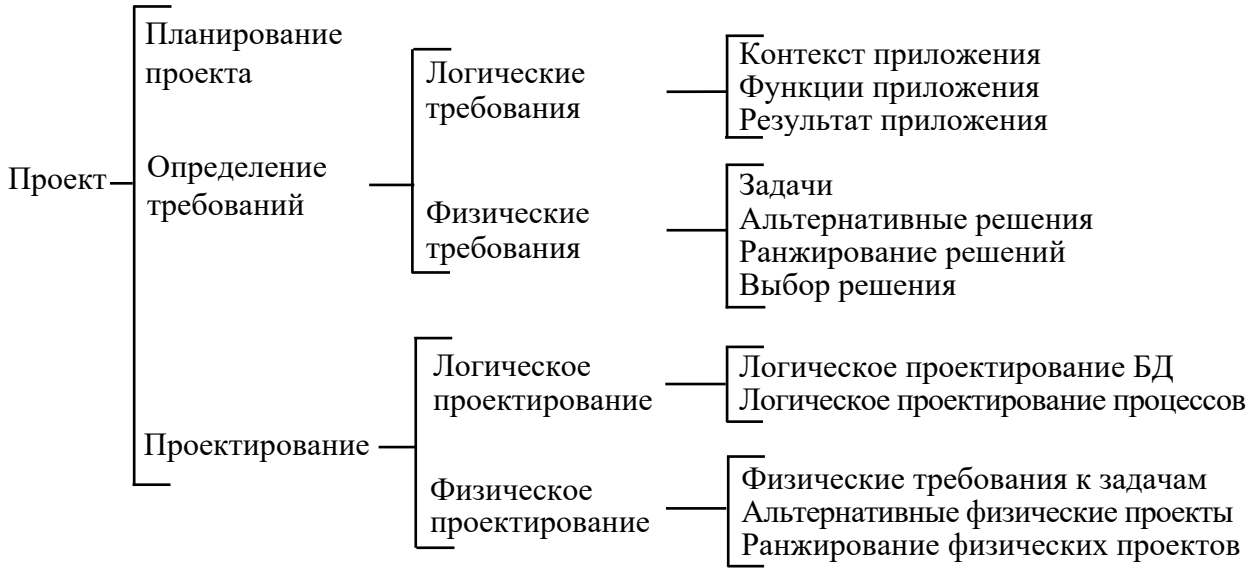


Рисунок 3.1 – Характеристика этапов методологии DSSD на диаграмме Варнье – Орра

При построении модели в DSSD используются диаграммы сущностей (диалект DFD) для определения системного контекста и диаграммы Варнье – Орра в качестве основного средства. Базовым элементом диаграммы Варнье – Орра является множественная скобка. Детализация элементов данных производится слева направо, предполагаемая последовательность действий осуществляется слева направо и сверху вниз. Такая нотация удобна для представления композиции структур, определения структур данных, спецификации форматов файлов. Она может быть использована для иллюстрирования структуры программы и иерархии модулей (заменой структур данных на модули или файлы, а на нижних уровнях – на подпрограммы, DO-циклы, условные и другие операторы), являясь в этом случае аналогом визуального языка проектирования.

IE-методология Мартина предоставляет собой общую стратегию разработки ИС, фокусирующую внимание на стратегическом планировании и бизнес-

процессах [1]. В то же время она является и инженерным подходом к разработке ПО, так как обеспечивает нисходящую пошаговую процедуру построения ИС. Основные этапы методологии Мартина приведены в таблице 3.4.

Таблица 3.4 – Этапы IE-методологии Мартина

Этап	Действие	Диаграмма
Стратегическое информационное планирование	Разработка стратегического плана для бизнес-системы, включающего цели и стратегии их достижения. Построение модели предметной области, отражающей существующую специфику и определяющей основные бизнес-процессы и организационную структуру бизнес-системы. Определение порядка разработки ИС	Диаграммы декомпозиции (для представления основных бизнес-процессов). Диаграммы «сущность-связь» (для представления основных структур данных). Матрицы информационного планирования
Анализ бизнес-процессов	Основные бизнес-процессы, описанные на предыдущем этапе, используются для разбиения общей задачи на частные. Разрабатываются информационная и функциональная модели для частных задач. При этом диаграммы «сущность-связь» преобразуются в нормализованную модель данных, а диаграммы декомпозиции распределяются по подзадачам	Для представления данных служат нормализованные модели данных. Для представления процессов служат DFD-диаграммы зависимости данных (диалект DFD) и диаграммы декомпозиции. Для соотнесения данных и процессов, использующих эти данные, – матрицы «сущность/процесс»
Логическое проектирование	Базой являются процессы, разработанные на этапе анализа. С помощью методики нисходящей функциональной декомпозиции проектируются спецификации обработки процессов и их логические структуры данных	Диаграммы структуры данных (диалект ERD), определяющие типы сущностей, атрибуты, связи. Диаграммы декомпозиции и диаграммы деятельности (вид мини спецификации), детализирующие логику процессов. Схемы экранов/отчетов, представляющие прототипы пользовательских интерфейсов
Физическое проектирование и реализация	Производится преобразование логической модели системы в физическую и ее реализация	Физическая модель системы. Программная система

Далее будут рассмотрены **методологии объектно-ориентированного анализа и проектирования**. **Объектно-ориентированный анализ** (OOA) – это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области. **Объектно-ориентированное проектирование** (OOD) – это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы. **Объектно-ориентированное программирование** (OOP) – это методо-

логия программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования [1].

Программа будет объектно-ориентированной только при соблюдении всех трех следующих требований:

1) объектно-ориентированное программирование использует в качестве базовых элементов объекты, а не алгоритмы (иерархия «быть частью»);

2) каждый объект является экземпляром какого-либо определенного класса;

3) классы организованы иерархически.

Так, программирование, не основанное на иерархических отношениях, не относится к ООР, а называется программированием на основе абстрактных типов данных.

Объектными принято называть языки, которые поддерживают абстракции данных и классы. **Объектно-ориентированными** являются те объектные языки, которые поддерживают наследование и полиморфизм. **Унифицированный язык моделирования** (UML – Unified Modeling Language) является языком, использующим графическую нотацию и предназначенный для спецификации, визуализации, конструирования и документирования систем ПО на основе объектно-ориентированных методов и компонентного подхода [11]. Использование универсального языка моделирования UML предусматривает построение следующих основных типов диаграмм:

– диаграммы вариантов использования;

– диаграммы классов;

– диаграммы состояний;

– три диаграммы поведения (две последние из них также называют диаграммами взаимодействия): диаграммы деятельности; диаграммы последовательности; диаграммы кооперации;

– две диаграммы реализации: диаграммы компонентов; диаграммы развертывания.

Методология проектирования ПО на базе языка UML называется **рациональный унифицированный процесс проектирования** (RUP – Rational Unified Process) [11]. Процесс – это частично упорядоченное множество шагов, направленных на достижение некоторой цели. RUP итеративен. В центре разработки в рамках этого процесса лежит архитектура. Основное внимание уделяется раннему определению архитектуры программного комплекса и формулированию основных ее особенностей. Фаза – это промежуток времени между двумя важными опорными точками процесса, в которых должны быть достигнуты четко определенные цели и принято решение о переходе к следующей фазе. Внутри каждой фазы происходит несколько итераций. Итерация представляет полный цикл разработки, от выработки требований во время анализа до реализации и тестирования. Все фазы и итерации подразумевают определенные затраты усилий на снижение рисков. В конце каждой фазы находится четко определенная опорная точка, где оценивается, в какой мере достигнуты намеченные цели и не следует ли внести в процесс изменения, прежде чем двигаться дальше. RUP состоит из четырех фаз (таблица 3.5).

Таблица 3.5 – Этапы IE-методологии Мартина

Фаза	Действие	Результат
Начало	Определяются цели системы и устанавливаются рамки проекта. Анализ целей включает выработку критерия успешности, оценку рисков, необходимых ресурсов и составление плана, в котором отражены основные опорные точки	Зачастую создается исполняемый прототип, демонстрирующий реалистичность концепции
Исследование	Задача – проанализировать предметную область, выработать прочные архитектурные основы, составить план проекта и устранить наиболее опасные риски. Для подтверждения правильности выбора архитектуры создается система, демонстрирующая выбранные принципы в действии и реализующая некоторые наиболее важные прецеденты	Диаграммы вариантов использования. Диаграммы классов. Диаграммы деятельности
Построение	В фазе построения постепенно и итеративно разрабатывается продукт, готовый к внедрению. Описываются оставшиеся требования и критерии приемки, проект «обрастает плотью», завершается разработка и тестирование программного комплекса. Продолжается итеративная работа с диаграммами классов и диаграммами деятельности	Строятся диаграммы взаимодействия: диаграммы последовательности; диаграммы кооперации. В случае сложного поведения системы разрабатываются диаграммы состояний
Внедрение	В фазе внедрения ПО передается пользователям. После этого решаются требующие дополнительной проработки вопросы по настройке системы, исправлению ошибок, ранее оставшихся незамеченными, и окончательному оформлению ряда функций, реализация которых была отложена	Обычно эта стадия воплощения проекта начинается с выпуска бета-версии системы, которая затем замещается коммерческой версией

Существует несколько рекомендаций, связанных с этой методологией: не следует стремиться к построению диаграмм всей системы; 80 % проектов можно воплотить, используя 20 % средств языка UML. Как правило, наиболее часто используют диаграммы классов и диаграммы деятельности.

Методология Шлеер – Меллора использует три группы средств для создания модели предметной области:

1) *информационное моделирование* – для определения отношений между данными (информацией). Применяются диаграммы классов;

2) *моделирование состояний* – для определения, зависящего от времени поведения системы. Используются диаграммы состояний;

3) *моделирование процессов* – для определения функций, которые система должна выполнить. Применяются диаграммы деятельности; диаграммы последовательности.

Для анализа больших предметных областей используются диаграммы, по смыслу близкие к следующим диаграммам языка UML: диаграммы кооперации; диаграммы компонентов; диаграммы развертывания. Методология Шлеер – Меллора предлагает механизм поддержки моделей состояний. Для этого используются четыре архитектурных класса:

1) переход – описывает каждый переход для всех моделей состояний программе;

2) конечная модель состояний – связывает все экземпляры перехода, которые составляют одну модель состояний;

3) активный экземпляр – это абстрактный класс, из которого все экземпляры, имеющие модель состояний, наследуют их текущее состояние;

4) таймер – обеспечивает механизм функционирования таймеров на основе аппаратных средств, доступных для хранения времени.

Методология Буча Г. [11] базируется на сочетании *микропроцесса* и *макропроцесса* разработки.

Микропроцесс объектно-ориентированной разработки приводится в движение потоком сценариев и продуктов архитектурного анализа (макропроцесс). Микропроцесс представляет ежедневную деятельность команды разработчиков и состоит из четырех шагов:

– идентификация классов и объектов на данном уровне абстракции; основными видами деятельности являются открытие и изобретение;

– выявление семантики классов и объектов; основными видами деятельности являются раскадровка сценариев, проектирование изолированных классов и поиск шаблонов;

– выявление связей между классами и объектами; основными действиями являются спецификация ассоциаций, выявление взаимодействий и уточнение ассоциаций;

– реализация классов и объектов; основное действие – выбор структур данных и алгоритмов.

Макропроцесс объектно-ориентированной разработки управляет микропроцессом, определяет измеримые характеристики проекта и помогает контролировать риск. Макропроцесс состоит из пяти шагов.

Концептуализация. Устанавливает основные требования к системе. Она служит для опробования концепций и, по большей части, не должна контролироваться, чтобы предоставить неограниченную свободу фантазии.

Анализ. Цель – получить модель поведения системы. Основные действия на этом этапе – анализ предметной области и планирование сценариев.

Проектирование. Создается архитектура реализации и вырабатываются единые тактические приемы. Основными действиями являются архитектурное планирование, тактическое проектирование и планирование релизов.

Эволюция. Последовательное приближение системы к желаемому результату. Основные действия – применение микропроцесса и управление изменениями.

Сопровождение. Управление эволюцией системы в ходе ее эксплуатации. Основные действия похожи на действия предыдущего шага, но к ним добавляется работа со списком улучшений и исправлений.

Далее рассмотрим промышленные методологии проектирования нескольких фирм-разработчиков ПО.

Методология DATARUN является распространенной электронной методологией. В соответствии с этой методологией ЖЦ ПО разбивается на стадии, которые связаны с результатами выполнения основных процессов, определяемых стандартом ISO/EEC 12207 [3]. Методология DATARUN опирается на две мо-

дели или на два представления: *модель организации* и *модель системы управления*. Основной принцип DATARUN заключается в том, что первичные данные, если они должным образом организованы в модель данных, становятся основой для проектирования архитектуры системы. Архитектура системы будет более стабильной, если она основана на первичных данных, тесно связанных с основными деловыми операциями, определяющими природу бизнеса, а не на традиционной функциональной модели. Методология преследует две цели:

1) определить стабильную структуру, на основе которой будет строиться система. Такой структурой является модель данных, полученная из первичных данных, представляющих фундаментальные процессы организации;

2) спроектировать систему на основе модели данных.

В процессе проектирования системы используется CASE-средство Silverrun, которое обеспечивает автоматизацию проведения проектных работ и последовательное построение следующего комплекса моделей:

– Business Process Model (BPM) – модель бизнес-процессов;
– Primary Data Structure (PDS) – структура первичных данных;
– Conceptual Data Model (CDM) – концептуальная модель данных;
– System Process Model (SPM) – модель процессов системы;
– Information System Architecture (ISA) – архитектура системы;
– Application Data Model (ADM) – модель данных приложения;
– Interface Presentation Model (IPM) – модель представления интерфейса;
– Interface Specification Model (ISM) – модель спецификации интерфейса.

Процесс проектирования системы состоит из следующих стадий.

Создаваемая система должна основываться на функциях, выполняемых организацией. Поэтому первая создаваемая модель – это модель бизнес-процессов, построение которой осуществляется в модуле Silverrun BPM. В процессе анализа и спецификации бизнес-функций выявляются основные информационные объекты, которые документируются как структуры данных, связанные с потоками и хранилищами модели. В процессе обследования работы организации выявляются и документируются структуры первичных данных. Эти структуры заносятся в репозиторий модуля BPM при описании циркулирующих в организации документов, сообщений, данных. В модели бизнес-процессов первичные структуры данных связаны с потоками и хранилищами информации.

На основе структур первичных данных в модуле Silverrun ERX создается концептуальная модель данных (ER-модель). От структур первичных данных концептуальная модель отличается удалением избыточности, стандартизацией наименований понятий и нормализацией. Эти операции в модуле ERX выполняются при помощи встроенной экспертной системы. Цель концептуальной модели данных – описать используемую информацию без деталей возможной реализации в базе данных, но в хорошо структурированном нормализованном виде.

На основе модели бизнес-процессов и концептуальной модели данных проектируется архитектура системы. Определяются входящие в систему приложения, для каждого приложения специфицируются используемые данные и реализуемые функции. Архитектура системы создается в модуле Silverrun BPM с использованием специальной нотации ISA. Основное содержание этой модели –

структурные компоненты системы и навигация между ними. Концептуальная модель данных разбивается на части, соответствующие входящим в состав системы приложениям.

Перед разработкой приложений должна быть спроектирована структура базы данных. Методология DATARUN предполагает использование базы данных, основанной на реляционной модели. Концептуальная модель данных после нормализации переносится в модуль реляционного моделирования Silverrun RDM с помощью специального моста ERX<–>RDM. После преобразования ERX в RDM получается модель реляционной базы данных. Эта модель детализируется в модуле Silverrun RDM определением физической реализации (типов данных СУБД, ключей, индексов, триггеров, ограничений ссылочной целостности).

С помощью модели системных процессов детально документируется поведение каждого приложения. В модуле BPM создается модель системных процессов, определяющая, как реализуются бизнес-процессы. Эта модель создается отдельно для каждого приложения и тесно связана с моделью данных приложения. Приложение состоит из интерфейсных объектов (экранных форм, отчетов, процедур обработки данных). Каждый интерфейсный объект связан с некоторым подмножеством базы данных. В модели данных приложения формируется подсхема базы данных для каждого интерфейсного объекта этого приложения.

Модель представления интерфейса – это описание внешнего вида интерфейса, как его видит конечный пользователь системы. Это может быть и документ, показывающий внешний вид экрана или структуру отчета, и сам экран, созданный с помощью средств визуальной разработки приложений.

После создания подсхем реляционной модели для приложений проектируется детальная структура каждого приложения в виде схемы навигации экранов, отчетов, процедур пакетной обработки. Полученная модель детально документирует приложение и непосредственно используется для программирования специфицированных интерфейсов.

Далее средствами разработки приложений происходит физическое создание системы: приложения программируются и интегрируются в ИС.

Методология фирмы ORACLE обеспечивает поддержку полного ЖЦ ПО для систем, использующих СУБД ORACLE, и базируется на интегрированном CASE-средстве Designer в совокупности со средствами разработки приложений Developer. Базовая методология фирмы ORACLE (CASE Method) – это структурная методология проектирования систем, охватывающая полностью все этапы ЖЦ ПО. Методология состоит из следующих этапов.

На этапе планирования определяются цели создания системы, приоритеты и ограничения, разрабатывается системная архитектура и составляется план разработки системы. В процессе анализа строятся модель информационных потребностей (диаграмма «сущность – связь»), диаграмма функциональной иерархии (на основе функциональной декомпозиции системы), матрица перекрестных ссылок и диаграмма потоков данных.

На этапе проектирования разрабатывается подробная архитектура системы, проектируются схема реляционной базы данных и программные модули, устанавливаются перекрестные ссылки между компонентами системы для анализа их

взаимного влияния и контроля за изменениями.

На этапе реализации создается база данных, строятся прикладные подсистемы, производятся их тестирование, проверка качества и соответствия требованиям пользователей. Создается системная документация, материалы для обучения и руководства пользователей.

На этапах эксплуатации и сопровождения анализируются производительность и целостность системы, выполняется поддержка и при необходимости модификация системы.

Методология фирмы Microsoft является моделью процессов (Microsoft Solution Framework – MSF) и представляет собой технологический подход, базирующийся на наборе моделей, правил и спецификаций, применяемых при создании и распространении программных продуктов. Модель процессов MSF возникла на основе используемого в компании Microsoft подхода к разработке ПО. В результате своего развития она объединила ряд наиболее эффективных принципов других известных моделей. Фазы модели процессов MSF приведены в таблице 3.6.

Таблица 3.6 – Фазы модели процессов MSF

Фаза	Действие
Выработка концепции	Создание и сплочение проектной группы на основе выработки единого видения. Выработка высокоуровневого взгляда на цели и условия проекта может рассматриваться как ранняя форма планирования, она подготавливает почву для процессов создания детальных планов, которые будут осуществлены непосредственно во время фазы планирования
Планирование	Производится основная работа по составлению планов проекта. Она включает в себя подготовку проектной группой функциональной спецификации, разработку дизайнов, подготовку рабочих планов, оценку проектных затрат и сроков разработки различных составляющих проекта
Разработка	Проектная группа фокусируется на создании компонент решения (включая как документацию, так и программный код). Однако некоторая часть этой работы может продолжаться также на фазе стабилизации, если такая необходимость выявлена в процессе тестирования. Данная фаза также включает в себя разработку инфраструктуры
Стабилизация	Производится тестирование разработанного решения. При этом внимание фокусируется на его эксплуатации в реалистичной модели производственной среды. Проектная группа занимается определением приоритета ошибок и их устранением, а также подготовкой решения к выпуску
Внедрение	Проектная группа внедряет технологии и компоненты решения, стабилизирует внедренное решение, передает работу персоналу поддержки и сопровождения и получает со стороны заказчика окончательное одобрение результатов проекта. По завершении внедрения проводится анализ выполненной работы

Тремя особенностями MSF являются:

1) *подход, основанный на фазах и вехах*. Вехи используются как опорные точки для планирования и мониторинга хода проекта. *Главные вехи* – это моменты ЖЦ проекта, когда полученные на той или иной фазе результаты синхронизируются членами проектной группы друг с другом и с ожиданиями заказчика.

В этот момент заказчиком, заинтересованными сторонами и проектной группой проводится формальный анализ достигнутого прогресса и достигается согласие продолжать далее работу над проектом;

2) *итеративный подход*. Программный код, документация, дизайн, планы и другие рабочие материалы создаются, как правило, итеративными методами. MSF рекомендует начинать разработку решения с построения, тестирования и внедрения его базовой функциональности. Далее используется стратегия версионирования – к решению добавляются все новые и новые возможности;

3) *интегрированный подход к созданию и внедрению решений*. Модель процессов MSF содержит весь ЖЦ создания решения, включая его внедрение.

Задание

Обосновать для проектируемой ИС выбор методологии и описать основные этапы проектирования с ее помощью.

Содержание отчета: титульный лист; тема и цель работы; текст индивидуального задания; выполнение индивидуального задания.

Контрольные вопросы

1 Перечислить три группы методологий проектирования программных систем.

2 Охарактеризовать методологию Йордона, методологию Гейна и Сарсона, методологию SSADM, методологию DSSD, IE-методологию Мартина, методологии объектно-ориентированного анализа и проектирования (OOA, OOD, OOP), рациональный унифицированный процесс проектирования, методологию Шлеер – Меллора, методологию Буча Г.

3 Охарактеризовать промышленные методологии проектирования DATARUN, фирмы ORACLE, Microsoft Solution Framework (MSF).

4 Выделить функциональные компоненты ИС. Определить компоненты представления программной системы. Охарактеризовать прикладной компонент и компонент управления ресурсами программной системы.

4 Анализ требований и разработка внешних спецификаций

Цель работы: составить спецификацию требований к разрабатываемой ИС.

Краткие теоретические сведения

Спецификация требований. Требования необходимо специфицировать (т. е. задать) графически или каким-либо иным формальным способом. Всесторонняя спецификация системы может потребовать использования различных типов моделей [2, 12]. Язык UML включает различные интегрированные методы моделирования, способные помочь бизнес-аналитику справиться с этой задачей. Спецификация – подобно процессу разработки ПО в целом – итеративный процесс с

пошаговым наращиванием уровня детализации моделей. Важную роль в моделировании играет использование CASE-средств.

Принципы спецификации требований. Спецификация требований связана с детальным моделированием требований заказчика, определенных в процессе установления требований. Рассматриваются только услуги, которые стремится получить от системы заказчик (формулировки сервисов). На этапе спецификации требований формулировки ограничений не подлежат дальнейшей проработке, хотя и могут претерпеть изменения как результат обычного цикла итерации.

В качестве входной информации процесса спецификации требований выступают неформальные требования заказчиков, а результатом этого процесса являются модели спецификации проектных конструкций. Эти модели дают более формальное определение различных сторон (представлений) системы. Обычно требования пользователей в процессе спецификации подразделяются на две основные категории: *функциональные требования* и *требования к данным*.

Результатом этапа спецификации является расширенный («детально проработанный») *документ описания требований*. Этот документ часто называют *документом спецификации требований* (или просто «спецификацией»). Структура исходного документа не изменяется, но содержание *значительно расширяется* за счет глав, определяющих требования заказчиков. Постепенно для целей проектирования и реализации документ спецификации требований заменяет документ описания требований (на практике расширенный документ может по-прежнему называться документом описания требований).

В результате спецификации требований разрабатываются три категории моделей спецификаций, представляемые в виде диаграмм на языке UML:

1) *модели состояний* – «детализируют» требования к данным;

2) *модели поведения* – обеспечивают детализированные спецификации для функциональных требований;

3) *модели изменения состояния* – охватывают два вида требований и призваны объяснить, как действие функций приводит к изменению данных.

Обычно диаграмма служит целям моделирования одной из сторон системы – состояний, поведения или изменения состояний. Исключение составляет диаграмма классов, которая определяет все три аспекта – состояние и поведение объектов, и, косвенно, изменения состояний объектов.

Каждая диаграмма дает представление об определенной стороне системы. Взятые вместе диаграммы дают возможность разработчикам и пользователям взглянуть на предлагаемое решение с разных точек зрения, выделяя одни его стороны и игнорируя другие. Ни одна из диаграмм в отдельности не дает полного определения системы. Полное представление о системе можно получить только через взаимосвязанный набор диаграмм.

Разработка диаграмм – это не последовательный процесс построения одной диаграммы за другой. Если разработчики должны следовать определенному процессу разработки, то решение о том, какая из моделей должна играть роль «движущей силы» разработки, в значительной степени зависит от личных предпочтений аналитика. Чаще всего диаграммы прецедентов и модели классов (как наиболее важные типы моделей) конструируются параллельно, взаимно «обогащая»

друг друга подробностями.

С каждой новой итерацией разработки глубина и степень детализации спецификации возрастает. Многие более глубокие свойства объектов модели выражаются скорее в текстовом, нежели графическом виде. Некоторые свойства определяют замысел объекта модели, а не результат анализа. Некоторые другие свойства могут отражать особенности CASE-средств.

Спецификации состояний. Состояние объекта определяется значениями его атрибутов и ассоциаций. Так, объект BankAccount (Банковский счет) может быть в состоянии «превышение кредитного лимита», если атрибут balance (баланс) имеет отрицательное значение [2, 12]. Так как состояние объекта определяется структурой данных, модели структур данных называются *спецификациями состояний*. Спецификация состояний обеспечивает статический взгляд на систему (поэтому моделирование состояний иногда называют статическим моделированием). Основная задача здесь – определение классов проблемной области, их атрибутов и отношений с другими классами. Изначально операции классов, как правило, не рассматриваются, а выводятся из моделей спецификации поведения.

Сначала определяются **классы-сущности**, т. е. классы, которые определяют проблемную область и характеризуются постоянным присутствием в базе данных системы. Такие классы иногда называют «бизнес-классами».

Управляющие классы (которые обслуживают системные события) и **классы представления** или **пограничные классы** (которые представляют GUI-интерфейс) устанавливаются только тогда, когда станут известны поведенческие характеристики системы.

Пример спецификации требований для информационной системы «Запись на университетские курсы» [12]. Университет проводит набор студентов и аспирантов дневной и вечерней форм обучения для подготовки по ряду специальностей. Университет состоит из факультетов, и каждый факультет имеет в своем составе несколько кафедр. Обучение по определенной специальности может включать дисциплины, преподаваемые на разных факультетах. Студенты могут выбирать часть курсов (т. е. учебных дисциплин), изучаемых для получения специальности, с учетом определенных ограничений (например, несоответствие расписаний, максимальная вместимостью аудиторий и т. д.).

Проектируемая система должна сопровождать процесс записи на курсы. Предварительная деятельность по записи должна включать рассылку оценок последнего семестра студентам наряду с инструкциями по записи на лекции. В период записи на курсы система должна принимать заявленную студентом последовательность прохождения курсов и проверять ее на предмет обязательности, конфликтов расписания, вместимости аудиторий, специальных санкций и т. д. Решения по некоторым проблемам могут требовать консультаций с преподавателями, ответственными за дисциплины.

Выявление классов. Для выявления классов могут использоваться следующие подходы [12], представленные в таблице 4.1.

Таблица 4.1 – Подходы к выявлению классов

Подход	Суть	Вид выделяемых классов
На основе использования именных групп (noun phrase approach)	В техническом задании определяются основные имена сущностей – имена существительные. Каждое имя существительное рассматривается как **потенциальный класс** (candidate class). Список всех классов делится на три группы	**Нерелевантные** (irrelevant) классы – классы, выходящие за рамки проблемной области (их предназначение не поддается описанию). Нерелевантные классы могут не включаться в список потенциальных классов. **Релевантные** (relevant) классы безусловно принадлежат проблемной области. Имена существительные, представляющие имена этих классов, часто встречаются в техническом задании. **Нечеткие** (fuzzy) классы нельзя безоговорочно признать релевантными. Их нужно глубоко проанализировать, а затем либо включить в список релевантных классов, либо исключить из списка нерелевантных
На основе использования общих шаблонов для классов (common class patterns approach)	Позволяет вывести **потенциальные классы** на основе теории родовой классификации объектов. Теория классификации описывает методы разделения мира объектов на группы с целью более подробного изучения. Этот подход можно использовать для выявления начального множества классов	Дж. Рамбо [12] и соавторы предложили, например, следующую схему классификации: физический класс (physical class), например Airplane (Самолет) бизнес-класс (business class), например Reservation (Резервирование) логический класс (logical class), например FlightTimetable (Расписание рейсов) прикладной класс (application class), например ReservationTransaction (Операция резервирования) компьютерный класс (computer class), например Index (Индекс) поведенческий класс (behavioral class), например ReservationCancellation (Отмена резервирования)
На основе использования прецедентов	Диаграмма прецедентов использования сопровождается неформальным описанием, а также диаграммами деятельности и взаимодействий. Эти дополнительные описания и модели определяют этапы (и объекты), необходимые в каждом прецеденте использования	После выявления прецедентов и частичного определения моделей взаимодействия объекты, используемые в этих диаграммах, приводят к идентификации **потенциальных классов**. Данный подход страдает от тех же недостатков, что и подход на основе именных групп. Будучи по сути подходом снизу вверх, он обеспечивает точность описания за счет полноты и корректности моделей прецедентов. *Частичные модели прецедентов* использования приводят к *неполным моделям классов*. Более того, модели классов соответствуют определенным функциональным свойствам и не всегда отражают все важные понятия предметной области
Подход CRC	Подход CRC: class – responsibility – collaborators (класс – обязанности – «сотрудники»). Включает в себя сеансы «мозгового штурма», проведение которых облегчается за счет использования специально подготовленных карточек	Карточки состоят из трех разделов: в ходе исполнения сценария обработки информации имя класса записывается в верхнем разделе, обязанности класса перечислены в левом разделе, а сотрудники перечислены в правом. **Обязанности** – это услуги (операции), которые класс готов выполнить в интересах других классов. Для выполнения многих обязанностей необходимо **участие (обслуживание)** со стороны других **классов-сотрудников**

Окончание таблицы 4.1

Подход	Суть	Вид выделяемых классов
		Подход CRC идентифицирует классы и их свойства на основе анализа передачи сообщений между объектами при выполнении задач обработки информации
Комплексный подход (mixed approach)	Включает элементы всех четырех подходов, рассмотренных выше. Важными факторами при этом выступают общая эрудиция эксперта, его опыт и интуиция. Процесс идет ни снизу вверх, ни сверху вниз – он все время идет «из середины»	Один из возможных сценариев заключается в следующем. **Начальное множество классов** можно сформировать на основе общих знаний и опыта эксперта. При этом дополнительно можно руководствоваться *подходом на основе использования общих шаблонов*. Остальные классы можно добавить, основываясь на анализе обобщенного описания проблемной области с использованием *подхода на основе использования именных групп*. Если в распоряжении аналитика имеются прецеденты использования системы, то можно воспользоваться *подходом на основе использования прецедентов*, чтобы добавить новые и проверить существующие классы. Наконец, *подход CRC* позволяет применить «мозговой штурм» для проверки правильности списка выявленных классов

Некоторые **правила выявления классов**, которым должен следовать аналитик при выборе *потенциальных классов* на роль *классов-сущностей* [12]:

– для каждого класса-сущности должно быть четко сформулировано его назначение в системе;

– каждый класс-сущность – это шаблон описания множества объектов;

– каждый класс-сущность должен содержать набор атрибутов. Хорошим приемом является установление идентифицирующих атрибутов (ключей), чтобы помочь судить о мощности (cardinality) класса (т. е. ожидаемом количестве объектов данного класса в базе данных). Однако, класс-сущность не обязательно должен обладать ключом;

– каждый класс-сущность содержит набор операций. Однако на данном этапе мы не касаемся вопросов идентификации операций. Операции, входящие в интерфейс класса (сервисы, предоставляемые классом системе), являются логическим следствием формулировки назначения класса;

– каждый класс-сущность должен отличаться от атрибута. Представляется ли понятие классом или атрибутом, зависит от области приложения. Цвет автомобиля обычно воспринимается как атрибут класса Car (Автомобиль). Однако на фабрике по производству красок Color (Цвет) – это определенно класс со своими собственными атрибутами (яркостью, насыщенностью, прозрачностью и т. д.);

– единичные классы (singleton classes), для которых можно представить существование только одного объекта, весьма маловероятны среди «бизнес-объектов». Подобные классы обычно составляют в приложении «общее знание» и, как правило, жестко запрограммированы в программах приложения. Например, если система спроектирована для единственной организации, существование класса Organization (Организация) может быть не оправдано.

Пример выявления классов. Рассмотрим следующие пять требований к системе «Запись на университетские курсы» и выделим *потенциальные классы*.

Для получения каждой университетской степени существует несколько обязательных и несколько выборочных курсов.

Каждому курсу соответствует заданный уровень и значение условных очков (CreditPoint – условное очко, начисляемое за прослушивание какого-либо курса (за один курс может быть начислено несколько очков), студент обязан на одном году набрать такое число курсов, чтобы число очков за них было не ниже определенного значения).

Курс может быть составной частью системы получения произвольного количества степеней.

Каждая степень определяет минимальное общее значение условных очков, требуемое для получения степени (например, для степени бакалавра (ИС) требуется 70 очков, включая обязательные курсы).

Студенты могут составлять из дисциплин программы обучения, приспособленные к их индивидуальным нуждам и обеспечивающие им получение степени, на которую они претендуют.

Проанализируем эти пять требований с целью выделения *потенциальных классов*. В первом утверждении подходящими классами являются классы Degree (Степень) и Course (Курс). Эти два класса удовлетворяют пяти правилам, перечисленным выше.

Пока неизвестно, должен ли и каким образом класс Course быть сужен до классов CompulsoryCourse (Обязательный курс) и ElectiveCourse (Выборочный курс). Например, ясно, что курс является обязательным или выборочным в зависимости от степени. Возможно, что различие между обязательными и выборочными курсами может быть зафиксировано с помощью ассоциации или даже атрибута класса. Таким образом, CompulsoryCourse и ElectiveCourse рассматриваются как нечеткие классы.

Второе утверждение идентифицирует только атрибуты класса Course, а именно course_level (уровень курса) и credit_point_value (количество условных очков).

Третье утверждение характеризует ассоциацию между классами Course и Degree.

Четвертая формулировка вводит атрибут min_total_credit_points (минимальное общее количество условных очков) в качестве атрибута класса Degree.

Последнее утверждение позволяет нам выделить три новых класса: Student (Студент), CourseOffering (Предлагаемый курс) и StudyProgram (Программа обучения). Первые два, безусловно, являются релевантными классами, а вот StudyProgram можно превратить в ассоциацию между классами Student и CourseOffering. Поэтому StudyProgram классифицируется как нечеткий класс.

Структура документа описания требований, получаемого на этапе установления требований, описана в подразделе 1.1 лабораторной работы № 1.

Формирование моделей спецификации (моделей состояний; моделей поведений; моделей изменения состояний) подробно описано в [1, 2, 12].

Построение диаграмм вариантов использования системы, деятельности, взаимодействия объектов подробно описано в [1, 2, 6, 8, 11, 12].

Задание

Составить спецификацию установленных в лабораторной работе № 1 требований для проектируемой ИС. Для составления спецификации использовать язык UML. Модели спецификации разделить на три группы: модели состояний; модели поведений; модели изменения состояний.

Построить UML-диаграммы к каждой группе моделей проектируемой системы и написать комментарии к ним. Нужно построить следующие диаграммы: вариантов использования системы, деятельности, взаимодействия объектов.

Содержание отчета: титульный лист; тема и цель работы; текст индивидуального задания; выполнение индивидуального задания.

Контрольные вопросы

1 Что такое спецификация требований?
2 Перечислить принципы спецификации требований.
3 Перечислить и охарактеризовать модели спецификаций.
4 Перечислить и охарактеризовать подходы к выявлению классов, а также правила выявления классов.

5 Анализ требований и определение спецификаций при объектном подходе

Цель работы: составить спецификацию требований к разрабатываемой ИС на основе объектного подхода.

Краткие теоретические сведения

При **объектном подходе** модели разрабатываемого ПО создаются на основе предметов и явлений реального мира. Модели базируются на описании требуемого поведения разрабатываемого ПО, т. е. его функциональности. Это поведение связывается с состояниями элементов (объектов) конкретной предметной области [12]. Следовательно, при выполнении анализа решаются две задачи:

1) сформулировать и уточнить требуемое поведение разрабатываемого ПО;
2) построить концептуальную модель его предметной области с точки зрения поставленных задач.

Объектный подход к разработке ПО основан на объектной декомпозиции, которая позволяет представить проектируемое ПО в виде совокупности объектов, при взаимодействии которых происходит выполнение требуемых функций посредством передачи сообщений.

UML является языком описания ПО с использованием **объектного подхода**. Для описания разработки на UML используются модели, характеризующие различные аспекты проектируемой ИС, представленные в таблице 5.1.

Таблица 5.1 – Подходы к выявлению классов

Наименование модели	Характеристика
Модель использования	Описывает функциональность ПО с точки зрения пользователя
Логическая модель	Определяет ключевые абстракции ПО (классы, интерфейсы и т. д.), т. е. средства, обеспечивающие нужную функциональность
Модель процессов	Отображает организацию вычислений и позволяет оценить производительность, масштабируемость и надежность ПО
Модель реализации	Устанавливает реальную организацию программных модулей
Модель развертывания	Показывает особенности размещения программных компонентов на конкретном оборудовании

UML предлагает девять дополняющих друг друга диаграмм, входящих в различные модели: диаграммы вариантов использования; диаграммы классов; диаграммы пакетов; диаграммы последовательностей действий; диаграммы кооперации; диаграммы деятельности; диаграммы состояний объектов; диаграммы компонентов; диаграммы размещения.

Полная спецификация разрабатываемого ПО, кроме указанных диаграмм включает словарь терминов, различные описания и текстовые спецификации.

Основным звеном объектно-ориентированных методов разработки ПО являются **диаграммы классов**. Однако диаграммы классов в этих методах в основном применяют на этапе проектирования для того, чтобы показать особенности построения конкретных классов. UML позволяет использовать три уровня диаграмм классов в зависимости от степени их детализации:

1) *концептуальный уровень*, на котором диаграммы классов, называемые в этом случае контекстными, показывают связи между основными понятиями предметной области. Модели на этом уровне оперируют понятиями предметной области, атрибутами этих понятий и отношениями между ними. Для выделения классов формируют множество понятий-кандидатов из существительных, характеризующих предметную область в описании вариантов использования;

2) *уровень спецификаций*, на котором диаграммы классов отображают интерфейсы классов предметной области, т. е. связи объектов этих классов;

3) *уровень реализации*, на котором диаграммы классов непосредственно показывают поля и методы конкретных классов.

Практически это уровни с тремя разными моделями, связь между которыми неоднозначна. Так, если модель концептуального уровня определяет некоторое понятие предметной области как класс, то это не означает, что для реализации этого понятия будет использован отдельный класс. Однако во всех трех моделях нас интересуют типы объектов (классы) и их статические отношения, что позволяет использовать единую нотацию.

Модель каждого из перечисленных уровней используется на конкретном этапе разработки ПО:

– на этапе *анализа* применяется диаграмма классов концептуального уровня;
– на этапе *проектирования* – диаграммы классов уровня спецификации;
– на этапе *реализации* – диаграммы классов уровня реализации.
Основные элементы диаграммы классов показаны в таблице 5.2.

Таблица 5.2 – Основные элементы диаграммы классов

Наименование	Назначение
Пакет (package)	Сущность, используемая для семантической группировки других сущностей
Класс (class)	Шаблон, абстрактное описание или представление свойств множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. В секциях класса указываются имя класса, атрибуты (отдельные свойства или признаки, которые являются общими для всех объектов данного класса) и операции (методы)
Ассоциация (association)	Указывает на наличие связи между экземплярами классов или объектами, например, класс «студент» ассоциирован с классом «университет». Ассоциация может иметь имя, например, «Учится». Рядом с именем ассоциации обычно ставят стрелку, указывающую направление чтения имени («Студент учится в университете», а не наоборот)
Обобщение (generalization)	Отношение между классами, при котором любой объект одного класса (подтипа) обязательно является также и объектом другого класса, называемого в данном контексте супертипом
Мощность связи (multiplicity)	Мощность (кратность) связи – это количество объектов, принимающих участие в связи. Мощность указывается на каждом конце связи. Кратность можно задать равной единице (1), указать диапазон: «ноль или единица» (0..1), «много» (0..*), «единица или больше» (1..*). Разрешается указывать определенное число (например, 5). Символ «*» обозначает произвольное конечное целое неотрицательное число, значение которого неизвестно на момент задания соответствующего отношения ассоциации

Связь между экземплярами классов подразумевает некоторые **роли**, которые соответствующие объекты играют по отношению друг к другу. Роль связана с направлением ассоциации. Так по отношению к студентам университет – организация, осуществляющая их обучение, т. е. роль университет можно назвать «Место учебы». Студент для университета – объект обучающей деятельности университета, т. е. «Обучаемый». Если роль собственного имени не имеет, то можно считать, что ее имя совпадает с именем класса, по отношению к которому определяется эта роль. Роль также обладает характеристикой множественности, которая показывает, сколько объектов может участвовать в одной связи с каждой стороны. Допускается указывать множественность: * – от 0 до бесконечности; <целое>.. * – от заданного числа до бесконечности; <целое> – точно определенное количество объектов; <целое1>, <целое2> – несколько вариантов точного количества объектов; <целое1>..<целое2> – диапазон объектов.

В [1, 2, 6, 8, 11, 12] подробно изложены особенности построения диаграмм классов, кооперации, деятельности, компонентов, размещения.

Задание

Составить для проектируемой ИС следующие диаграммы: диаграммы классов; диаграммы кооперации; диаграммы деятельности; диаграммы компонентов; диаграммы размещения.

Содержание отчета: титульный лист; тема и цель работы; текст индивидуального задания; выполнение индивидуального задания.

Контрольные вопросы

1 Что такое объектная модель?
2 Охарактеризовать уровни диаграмм классов.
3 Охарактеризовать основные типы диаграмм в UML.
4 Как строятся диаграммы классов; диаграммы кооперации; диаграммы деятельности; диаграммы компонентов; диаграммы размещения?

Список литературы

1 **Орлов, С. А.** Программная инженерия : учебник для вузов. Стандарт третьего поколения / С. А. Орлов. – Санкт-Петербург : Питер, 2016. – 640 с.

2 **Лионг, Б.** Практическая программная инженерия на основе учебного примера / Б. Лионг, Л. Мацяшек. – Москва : Бином, 2013. – 960 с.

3 **Гагарина, Л. Г.** Технология разработки ПО [Электронный ресурс]: учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул ; под ред. Л. Г. Гагариной. – Москва: ФОРУМ ; ИНФРА-М, 2022. – 400 с. – Режим доступа: https://znanium.com/catalog/ product/1699927. – Дата доступа: 05.12.2022.

4 **Назаров, С. В.** Архитектура и проектирование программных систем [Электронный ресурс] : монография / С. В. Назаров. – 2-е изд., перераб. и доп. – Москва : ИНФРА-М, 2020. – 374 с. – Режим доступа: https://znanium.com/ catalog/product/1093643. – Дата доступа: 05.12.2022.

5 **Рыбальченко, М. В.** Архитектура ИС: учебное пособие для вузов / М. В. Рыбальченко. – Москва : Юрайт, 2016. – 91 с.

6 **Арлоу, Д.** UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу. – Москва : Символ-Плюс, 2015. – 624 с.

7 **Макаровских, Т. А.** Документирование ПО. В помощь техническому писателю : учебное пособие / Т. А. Макаровских. – 2-е изд. – Москва : ЛЕНАНД, 2015. – 266 с.

8 **Dennis, A.** System Analysis & Design. An Object-Oriented Approach with UML = Системный анализ и проектирование на универсальном языке моделирования / A. Dennis, B. Wixom, D. Tegarden. – 5 th ed. – New York : John Wiley & Sons, 2015. – 544 p.

9 Проектирование архитектур ИС : методические указания к лабораторным работам / Сост. К. С. Беляев. – Ульяновск : УлГТУ, 2010. – 48 с.

10 **Коцюба, И. Ю.** Основы проектирования ИС : учебное пособие / И. Ю. Коцюба, А. В. Чунаев, А. Н. Шиков. – Санкт-Петербург : Университет ИТМО, 2015. – 206 с.

11 **Буч, Г.** Введение в UML от создателей языка: пер. с англ. / Г. Буч, Д. Рамбо, И. Якобсон. – 2-е изд. – Москва : ДМК Пресс, 2012. – 494 с. : ил.

12 **Мацяшек, Л. А.** Анализ и проектирование информационных систем с помощью UML 2.0: пер. с англ. / Л. А. Мацяшек. – 3-е изд. – Москва : Вильямс, 2008. – 816 с. : ил.