Министерство образования Иркутской области
Государственное бюджетное профессиональное образовательное учреждение
Иркутской области
«Иркутский авиационный техникум»
(ГБПОУИО «ИАТ»)

Рассмотрено
на заседании ВЦК ПКС
Протокол № 2 от 29.09.2017 г.

УТВЕРЖДАЮ
Директор ГБПОУИО «ИАТ»
А.Н. Якубовский
04.10.2017 г.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КУРСОВОГО ПРОЕКТА

ПМ.03 Участие в интеграции программных продуктов
МДК.03.01 Технология разработки программного обеспечения
для специальности
09.02.03 Программирование в компьютерных системах

Иркутск 2017

Содержание

1 Назначение и порядок выполнения курсового проекта

Целью выполнения курсового проекта является закрепление теоретических знаний, получаемых при изучении **МДК.03.01 Технология разработки программного обеспечения, МДК.02.02 Технология разработки и защиты баз данных**, **МДК.01.02 Прикладное программирование, МДК.03.02 Инструментальные средства разработки программного обеспечения, МДК.03.03 Документирование и сертификация**, и выработка навыков самостоятельной практической проектной работы.

Работа над курсовым проектом включает:

1) получение индивидуального задания;
2) анализ информационных источников;
3) разработка проекта приложения в соответствии с полученным заданием;
4) разработка рекомендаций по сопровождению проекта;
5) разработка рекомендаций по защите проекта;
6) определение способов поддержания целостности данных.

При работе над курсовым проектом необходимо выполнить следующие этапы:

Стадии	Этапы работ
1. Формирование требований к АИС	1.1. Обследование объекта и обоснование необходимости создания АИС 1.2. Формирование требований пользователя к АИС 1.3. Оформление отчета о выполненной работе и заявки на разработку АИС (тактико-технического задания)
2. Разработка концепции АИС	2.1. Изучение объекта 2.2. Проведение необходимых научно - исследовательских работ 2.3. Разработка вариантов концепции АИС и выбор варианта концепции АИС, удовлетворяющего требованиям пользователя 2.4. Оформление отчета о выполненной работе
3. Техническое задание	3.1. Разработка и утверждение технического задания на создание АИС

4. Эскизный проект	4.1. Разработка предварительных проектных решений по системе и ее частям 4.2. Разработка документации на АИС и ее части
5. Технический проект	5.1. Разработка проектных решений по системе и ее частям 5.2. Разработка документации на АИС и ее части 5.3. Разработка и оформление документации на поставку изделий для комплектования АИС и/или технических требований (технических заданий) на их разработку 5.4. Разработка заданий на проектирование в смежных частях проекта объекта автоматизации
6. Рабочая документация	6.1. Разработка рабочей документации на систему и ее части 6.2. Разработка или адаптация программ
7. Ввод в действие	7.1. Подготовка объекта автоматизации к вводу АС в действие 7.2. Подготовка персонала 7.3. Комплектация АС поставляемыми изделиями (программными и техническими средствами, программно-техническими комплексами, информационными изделиями) 7.4. Строительно-монтажные работы 7.5. Пусконаладочные работы 7.6. Проведение предварительных испытаний 7.7. Проведение опытной эксплуатации 7.8. Проведение приемочных испытаний
8. Сопровождение АИС	8.1. Выполнение работ в соответствии с гарантийными обязательствами 8.2. Послегарантийное обслуживание

В процессе выполнения этапов курсового проекта необходимо выполнить разработку алгоритмов и программ, реализующих эти алгоритмы. Все из разрабатываемых программ подлежат отладке на компьютере. Отчет по курсовому проекту должен быть стандартно оформлен, содержать анализ предметной области и исходные данные для проектирования, материалы по выполнению этапов проектирования, анализ вариантов организации базы данных. В отчете представляются таблицы, рисунки, наглядно поясняющие организацию данных, блок-схемы алгоритмов, исходные тексты программ,

контрольные распечатки данных. Отчет по курсовому проекту подлежит защите, по результатам которой выставляется дифференцированная оценка.

2 Состав курсового проекта

Курсовой проект состоит из аналитической и проектной части, выполняемой с применением компьютера. Аналитическая часть курсового проекта выполняется студентом на основании его знаний о предметной области, являющейся предметом проектирования, с привлечением имеющихся у него знаний о структуре, функционировании и документообороте реальных предприятий. Машинная реализация проекта производится с использованием СУБД (по выбору студента)

Работа над курсовым проектом является индивидуальной.

По согласованию с преподавателем допускается выполнение курсового проекта по индивидуальному заданию.

На основании технического задания на курсовой проект студент представляет себе и описывает объект, для которого проектируется информационная система.

Основываясь на этом представлении, студент должен описать результаты исследования на естественном языке и представить таблицы и числовые значения используемых в дальнейшем величин.

В материалах исследования должны содержаться данные о характере и масштабах деятельности объекта, его структуре. Привести обоснование перечня задач, которые необходимо решить для достижения цели проекта.

3 Выполнение курсового проекта по этапам

Состав процессов жизненного цикла регламентируется международным стандартом ISO/IEC 12207: 1995 «Information Technology Software Life-Cycle Processes» («Информационные технологии Процессы жизненного цикла программного обеспечения»).

Стандарт описывает структуру жизненного цикла программного обеспечения и его процессы.

Процесс разработки предусматривает работы:

− по созданию программного обеспечения и его компонентов в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации;

− подготовку материалов, необходимых для проверки работоспособности и соответствия качества программных продуктов, материалов, необходимых для обучения персонала.

Процесс разработки включает следующие действия:

• *подготовительную работу* - выбор модели жизненного цикла, стандартов, методов и средств разработки, а также составление плана работ;

• *анализ требований к системе* - определение ее функциональных возможностей, пользовательских требований, требований к надежности и безопасности, требований к внешним интерфейсам и т. д.;

• *проектирование архитектуры системы* - определение состава необходимого оборудования, программного обеспечения и операций, выполняемых обслуживающим персоналом;

• *анализ требований к программному обеспечению* - определение функциональных возможностей, включая характеристики производительности, среды функционирования компонентов, внешних интерфейсов, спецификаций надежности и безопасности, эргономических требований, требований к используемым данным, установке, приемке, пользовательской документации, эксплуатации и сопровождению;

• *проектирование архитектуры программного обеспечения* - определение структуры программного обеспечения, документирование интерфейсов его компонентов, разработку предварительной версии пользовательской документации, а также требований к тестам и плана интеграции;

• *детальное проектирование программного обеспечения* - подробное описание компонентов программного обеспечения и интерфейсов между ними, обновление пользовательской документации, разработка и документирование требований к тестам и плана тестирования компонентов программного обеспечения, обновление плана интеграции компонентов;

• *кодирование и тестирование программного обеспечения* - разработку и документирование каждого компонента, а также совокупности тестовых процедур и данных для их тестирования, тестирование компонентов, обновление пользовательской документации, обновление плана интеграции программного обеспечения;

• *интеграцию программного обеспечения* - сборку программных компонентов в соответствии с планом интеграции и тестирование программного обеспечения на соответствие квалификационным требованиям, представляющих собой набор критериев или условий, которые необходимо выполнить, чтобы квалифицировать программный продукт, как соответствующий своим спецификациям и готовый к использованию в заданных условиях эксплуатации;

• *квалификационное тестирование программного обеспечения* - тестирование программного обеспечения в присутствии заказчика для демонстрации его соответствия требованиям и готовности к эксплуатации; при. этом проверяется также готовность и полнота технической и пользовательской документации;

• *интеграцию системы* - сборку всех компонентов системы, включая программное обеспечение и оборудование;

• *квалификационное тестирование системы* - тестирование системы на соответствие требованиям к ней и проверка оформления и полноты документации;

• *установку программного обеспечения* - установку программного обеспечения на оборудовании заказчика и проверку его работоспособности;

• *приемку программного обеспечения* - оценку результатов квалификационного тестирования программного обеспечения и системы в целом и документирование результатов оценки совместно с заказчиком, окончательную передачу программного обеспечения заказчику.

Указанные действия можно сгруппировать, условно выделив следующие основные этапы разработки программного обеспечения (в скобка» укачаны соответствующие *стадии разработки* по ГОСТ 19.102-77 «Стадии разработки»):

− постановка задачи (стадия «Техническое задание»);

− анализ требований и разработка спецификаций (стадия «Эскизный проект»);

− проектирование (стадия «Технический проект»);

− реализация (стадия «Рабочий проект»).

Традиционно разработка также включала этап *сопровождения* (началу этого этапа соответствует стадия «Внедрение» по ГОСТ). Однако по международному стандарту в соответствии с изменениями, происшедшими в индустрии разработки программного обеспечения, этот процесс рассматривается отдельно.

Постановка задачи. В процессе *постановки задачи* четко формулируют назначение программного обеспечения и основные требования к нему. Каждое требование представляет собой описание необходимого или желаемого свойства программного обеспечения. Различают *функциональные требования,* определяющие функции, которые должно выполнять разрабатываемое программное обеспечение, и *эксплуатационные требования,* декларирующие особенности его функционирования.

Требования к программному обеспечению, имеющему *прототипы,* обычно определяют по аналогии, учитывая структуру и характеристики уже существующего программного обеспечения. Для формулирования требований к программному обеспечению проводят специальные исследования, называемые предпроектными. В процессе исследований определяют разрешимость задачи, разрабатывают методы её решения (если они новые) и устанавливают наиболее существенные характеристики разрабатываемого программного обеспечения. Для выполнения предпроектных исследований, как правило, заключают договор на

выполнение научно-исследовательских работ. Этап постановки задачи заканчивается разработкой *технического задания,* фиксирующего принципиальные требования, и принятием основных проектных решений.

Анализ требований и определение спецификаций. *Спецификациями* называют точное формализованное описание функций и ограничений разрабатываемого программного обеспечения. Соответственно различают *функциональные* и *эксплуатационные* спецификации. Совокупность спецификаций представляет собой *общую* логическую модель проектируемого программного обеспечения.

Для получения спецификаций выполняют анализ требований технического задания, формулируют содержательную постановку задачи, выбирают математический аппарат формализации, строят модель предметной области, определяют подзадачи и выбирают или разрабатывают методы их решения. Часть спецификаций может быть определена в процессе предпроектных исследований и соответственно зафиксирована в техническом задании.

На этапе целесообразно сформировать тесты для поиска ошибок в проектируемом программном обеспечении, обязательно указать ожидаемые результаты.

Проектирование. Основной задачей этапа является определение *подробных* спецификаций разрабатываемого программного обеспечения. Процесс проектирования сложного программного обеспечения включает:

− проектирование общей структуры - определение основных компонентов и их взаимосвязей;

− декомпозицию компонентов и построение структурных иерархий в соответствии с рекомендациями блочно-иерархического подхода;

− проектирование компонентов.

Результатом проектирования является *детальная модель* разрабатываемого программного обеспечения вместе со спецификациями его компонентов всех уровней. Тип модели зависит от выбранного подхода (структурный, объектный или компонентный) и конкретной технологии проектирования. Процесс проектирования охватывает, как проектирование программ (подпрограмм) и определение взаимосвязей между ними, так и проектирование данных, с которыми взаимодействуют эти программы или подпрограммы.

Принято различать два аспекта проектирования:

− логическое проектирование, включающее проектные операции, которые не зависят от имеющихся технических и программных средств, составляющих среду функционирования программного продукта;

− физическое проектирование - привязка к конкретным техническим и программным средствам среды функционирования с учетом ограничений определенных в спецификациях.

Реализация - процесс поэтапного написания кодов программы на выбранном языке программирования (кодирование), их тестирование и отладку.

Сопровождение - процесс создания и внедрения новых версий программного продукта:

− необходимость исправления ошибок, выявленных в процессе эксплуатации предыдущих версий;

− необходимость совершенствования предыдущих версий (расширение состава выполняемых функций или повышение его производительности);

− изменение среды функционирования (появление новых технических средств и/или программных продуктов).

На этапе в программный продукт вносятся изменения, которые, могут потребовать пересмотра проектных решений, принятых на любом предыдущем этапе.

4 Определение требований к программному обеспечению и исходных данных для его проектирования

4.1 Предпроектные исследования предметной области

Целью предпроектных исследований является преобразование общих нечетких знаний о предназначении будущего программного обеспечения в сравнительно точные требования к нему.

Существуют два варианта неопределенности:

− неизвестны методы решения формулируемой задачи - такого типа неопределенности обычно возникают при решении научно-технических задач;

− неизвестна структура автоматизируемых информационных процессов - обычно встречается при построении автоматизированных систем управления предприятиями.

В первом случае во время предпроектных исследований определяют возможность решения поставленной задачи и методы, позволяющие получить требуемый результат.

Во втором случае определяют:

• структуру и взаимосвязи автоматизируемых информационных процессов;

• распределение функций между человеком и системой, а также между аппаратурой и программным обеспечением;

• функции программного обеспечения; внешние условия его функционирования и особенности его интерфейсов как с пользователями, так и при необходимости - с аппаратурой;

• требования к программным и информационным компонентам, необходимые аппаратные ресурсы, требования к базам данных и физические характеристики программных компонент.

Результаты предпроектных исследований предметной области используют в процессе разработки технического задания.

4.2 Разработка технического задания

Техническое задание представляет собой документ, в котором сформулированы основные цели разработки, требования к программному продукту, определены сроки и этапы разработки и регламентирован процесс приемно-сдаточных испытаний. В разработке технического задания участвуют как представители заказчика, так и представители исполнителя. В основе этого документа лежат исходные требования заказчика, анализ передовых достижений техники, результаты выполнения научно-исследовательских работ, предпроектных исследований, научного прогнозирования и т. п.

Основные факторы, определяющие характеристики разрабатываемого программного обеспечения, являются:

• исходные данные и требуемые результаты, которые определяют *функции* программы или системы;

• среда функционирования (программная и аппаратная) - может быть задана, а может выбираться для обеспечения параметров, указанных в техническом задании;

• возможное взаимодействие с другим программным обеспечением и/или специальными техническими средствами - также может быть определено, а может выбираться исходя из набора выполняемых функций.

Разработка технического задания выполняется в следующей последовательности:

- устанавливается набор выполняемых функций;
- устанавливается перечень и характеристики исходных данных;
- определяется перечень результатов проектирования;
- определяется характеристики и способы представления результатов;
- уточняются среда функционирования программного обеспечения:

а) конкретную комплектацию и параметры технических средств;

б) версию используемой операционной системы;

в) версии и параметры установленного программного обеспечения;

- регламентируются действия программы в случае сбоев оборудования и энергоснабжения.

На техническое задание существует стандарт ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению». В соответствии с этим стандартом техническое задание должно содержать следующие разделы:

• введение;
• основания для разработки;
• назначение разработки;
• требования к программе или программному изделию;
• требования к программной документации;
• технико-экономические показатели;
• стадии и этапы разработки;
• порядок контроля и приемки.

При необходимости допускается в техническое задание включать приложения.

Введение должно включать наименование и краткую характеристику области применения программы или программного продукта, а также объекта (например, системы), в котором предполагается их использовать. Основное назначение введения - продемонстрировать актуальность данной разработки и показать, какое место эта разработка занимает в ряду подобных.

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

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

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

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

Наиболее важным из перечисленных выше является подраздел *Требования к функциональным характеристикам.* В этом разделе должны быть перечислены выполняемые функции и описаны состав, характеристики и формы представления исходных данных и результатов. В этом же разделе при необходимости указывают критерии эффективности: максимально допустимое время ответа системы, максимальный объем используемой оперативной и/или внешней памяти и др.

В подразделе *Требования к надежности* указывают уровень надежности, который должен быть обеспечен разрабатываемой системой и время восстановления системы после сбоя. Для систем с обычными требованиями к

надежности в этом разделе иногда регламентируют действия разрабатываемого продукта по увеличению надежности результатов (контроль входной и выходной информации, создание резервных копий промежуточных результатов и т. п.).

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

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

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

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

В разделе *Технико-экономические показатели* рекомендуется указывать ориентировочную экономическую эффективность, предполагаемую годовую потребность и экономические преимущества по сравнению с существующими аналогами.

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

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

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

В зависимости от особенностей разрабатываемого продукта разрешается уточнять содержание разделов, т. е. использовать подразделы, вводить новые разделы или объединять их.

В случаях если какие-либо требования, предусмотренные техническим заданием, заказчик не предъявляет, следует в соответствующем месте указать «Требования не предъявляются».

4.3 Принципиальные решения начальных этапов проектирования

Этап процесса проектирования включает:

− выбор архитектуры программного обеспечения;

− выбор типа пользовательского интерфейса и технологии работы с документами;

− выбор подхода к разработке (структурного или объектного);

− выбор языка и среды программирования.

Выбор архитектуры программного обеспечения. Архитектура программного обеспечения определяется сложностью решаемых задач, степенью универсальности разрабатываемого программного обеспечения и числом пользователей, одновременно работающих с одной его копией. Различают:

− однопользовательскую архитектуру;

− многопользовательскую архитектуру.

Кроме того, в рамках однопользовательской архитектуры различают:

− программы;

− пакеты программ;

− программные комплексы;

− программные системы.

Многопользовательскую архитектуру реализуют системы, построенные по принципу «клиент-сервер».

Выбор типа пользовательского интерфейса. Различают четыре типа пользовательских интерфейсов:

− *примитивные*;

− *меню*;

− *со свободной навигацией*;

− *прямого манипулирования*.

Тип пользовательского интерфейса во многом определяет сложность и трудоемкость разработки, которые существенно возрастают в порядке перечисления типов. По последним данным до 80 % программного кода может реализовывать именно пользовательский интерфейс.

Кроме того, выбор типа интерфейса включает выбор *технологии работы с документами.* Различают две технологии:

− однодокументная;

− многодокументная.

Трудоемкость реализации многодокументных интерфейсов с использованием современных библиотек примерно на 3...5 % выше, чем первого.

Выбор подхода к разработке. Выбор интерфейса со свободной навигацией или прямого манипулирования, предполагает использование событийного программирования и объектного подхода. При этом в зависимости от сложности предметной области программное обеспечение может реализовываться как с использованием объектов и соответственно классов, так и чисто процедурно.

Примитивный интерфейс и интерфейс типа меню совместимы как со структурным, так и с объектным подходами к разработке.

Практика показывает, что объектный подход эффективен для разработки очень больших программных систем (более 100 000 операторов универсального языка программирования) и в тех случаях, когда объектная структура предметной области ярко выражена.

Во всех случаях выбор подхода остается за разработчиком.

Выбор языка программирования. Язык может быть определен:

− организацией, ведущей разработку;

− программистом, который по возможности всегда будет использовать хорошо знакомый язык;

− устоявшимся мнением.

Все существующие языки программирования можно разделить на следующие группы:

− универсальные языки высокого уровня;

− специализированные языки разработчика программного обеспечения;

− специализированные языки пользователя;

− языки низкого уровня.

В группе *универсальных языков высокого уровня* лидером на сегодня все еще является язык С (вместе с С++).

Альтернативой С и С++ является Pascal, компиляторы которого в силу четкого синтаксиса обнаруживают помимо синтаксических и большое количество семантических ошибок.

Кроме этих языков к группе универсальных принадлежат также Basic, Modula, Ada и некоторые другие. Каждый из указанных языков имеет свои особенности и соответственно свою область применения.

Специализированные языки разработчика используют для создания конкретных типов программного обеспечения. К ним относят:

− языки баз данных;

− языки создания сетевых приложений;

− языки создания систем искусственного интеллекта и т. д.

Специализированные языки пользователя обычно являются частью профессиональных сред пользователя, характеризуются узкой направленностью и разработчиками программного обеспечения не используются.

Языки низкого уровня позволяют осуществлять программирование практически на уровне машинных команд. При этом получают самые оптимальные, как с точки зрения времени выполнения, так и с точки зрения

объема необходимой памяти программы.

В настоящее время языки типа Ассемблера обычно используют:

− при написании сравнительно простых программ, взаимодействующих непосредственно с техническими средствами, например драйверов, поскольку в этом случае приходится кропотливо настраивать соответствующее оборудование, преимущества языков программирования высокого уровня становятся несущественными;

− в виде вставок в программы на языках высокого уровня, например, для ускорения преобразования данных в циклах с большим количеством повторений.

Выбор среды программирования. *Средой программирования* называют программный комплекс, который включает специализированный текстовый редактор, встроенные компилятор, компоновщик, отладчик, справочную систему и другие программы, использование которых упрощает процесс написания и отладки программ.

Наиболее часто используемыми являются визуальные среды Delphi, C++ Builder фирмы Borland (Inprise Corporation), Visual C++, Visual Basic фирмы Microsoft, Visual Ada фирмы IBM и др.

Выбор между этими средами в значительной степени определяться характером проекта.

Выбор или формирование стандартов разработки. Реальное применение любой технологии проектирования требует формирования или выбора ряда стандартов, которые должны соблюдаться всеми участниками проекта:

• стандарт проектирования;

• стандарт оформления проектной документации;

• стандарт интерфейса пользователя.

Стандарт проектирования должен определять:

− набор необходимых моделей (схем, диаграмм) на каждой стадии проектирования и степень их детализации;

− правила фиксации проектных решений на диаграммах, в том числе правила именования объектов и соглашения по терминологии, набор атрибутов для всех объектов и правила их заполнения на каждой стадии, правила оформления диаграмм, включая требования к форме и размерам объектов;

− требования к конфигурации рабочих мест разработчиков, включая настройки операционной системы и используемых CASE-средств;

− механизм обеспечения совместной работы над проектом, в том числе и правила интеграции подсистем проекта и анализа проектных решений на непротиворечивость.

Стандарт оформления проектной документации должен регламентировать:

− комплектность, состав и структуру документации на каждой стадии;

− требования к ее содержанию и оформлению;

− правила подготовки, рассмотрения, согласования и утверждения

документов.

Стандарт интерфейса пользователя должен определять:

− правила оформления экранов (шрифты и цветовую палитру), состав и расположение окон и элементов управления;

− правила пользования клавиатурой и мышью;

− правила оформления текстов помощи;

− перечень стандартных сообщений;

− правила обработки реакции пользователя.

Все описанные выше проектные решения существенно влияют на трудоемкость и сложность разработки. Только после их принятия следует переходить к анализу требований и разработке спецификаций проектируемого программного обеспечения.

5 Анализ требований и определение спецификаций программного обеспечения при структурном подходе

5.1 Спецификации программного обеспечения при структурном подходе

Спецификации представляют собой *полное* и *точное* описание функций и ограничений разрабатываемого программного обеспечения.

Функциональные спецификации описывают функции разрабатываемого программного обеспечения.

Эксплуатационные спецификации определяют требования к техническим средствам, надежности, информационной безопасности и т. д.

Точные спецификации можно определить, только разработав некоторую *формальную модель* разрабатываемого программного обеспечения.

Формальные модели, используемые на этапе определения спецификаций можно разделить на две группы:

− модели, *зависящие от подхода к разработке* (структурного или объектно-ориентированного);

− модели, *не зависящие* от него.

В рамках структурного подхода на этапе анализа и определения спецификаций используют три типа моделей:

− ориентированные на функции;

− ориентированные на данные;

− ориентированные на потоки данных.

Каждую модель целесообразно использовать для своего специфического класса программных разработок.

Диаграммы переходов состояний определяют основные аспекты поведения

программного обеспечения во времени.

Диаграммы потоков данных - направление и структуру потоков данных.

Концептуальные диаграммы классов - отношение между основными понятиями предметной области.

Поскольку разные модели описывают проектируемое программное обеспечение с разных сторон, рекомендуется использовать сразу несколько моделей и сопровождать их текстами: словарями, описаниями и т. п., которые поясняют соответствующие диаграммы.

Методологии *структурного анализа и проектирования,* основанные на моделировании потоков данных, обычно используют комплексное представление проектируемого программного обеспечения в виде совокупности моделей:

- *диаграмм потоков данных* (DFD - Data Flow Diagrams), описывающих взаимодействие источников и потребителей информации через процессы, которые должны быть реализованы в системе;
- *диаграмм «сущность-связь»* (ERD - Entity-Relationship Diagrams), описывающих базы данных разрабатываемой системы;
- *диаграмм переходов состояний* (STD - State Transition Diagrams), характеризующих поведение системы во времени;
- *спецификаций процессов*;
- *словаря терминов*.

Спецификации процессов. Спецификации процессов представляют в виде краткого текстового описания, схем алгоритмов, псевдокодов, Flow-форм или диаграмм Насси-Шнейдермана. Чаще всего используют псевдокоды.

Словарь терминов. Словарь терминов представляет собой краткое описание основных понятий, используемых при составлении спецификаций, содержащих:

- определение основных понятий предметной области;
- описание структур элементов данных, их типов и форматов;
- описание сокращений и условных обозначений.

Он предназначен для повышения степени понимания предметной области и исключения риска возникновения разногласий при обсуждении моделей между заказчиками и разработчиками.

Обычно описание термина в словаре выполняют по следующей схеме:

- термин;
- категория (понятие предметной области, элемент данных, условное обозначение и т. д.);
- краткое описание.

5.2 Методологии структурного анализа и проектирования

5.2.1 Диаграммы переходов состояний

Диаграмма переходов состояний является графической формой предоставления *конечного автомата* - математической абстракции, используемой для моделирования детерминированного поведения технических объектов или объектов реального мира.

На этапе анализа требований и определения спецификаций диаграмма переходов состояний демонстрирует *поведение* разрабатываемой программной системы при получении управляющих воздействий.

Для построения диаграммы переходов состояний необходимо в соответствии с теорией конечных автоматов определить:

- основные состояния, управляющие воздействия (или условия перехода);
- выполняемые действия;
- возможные варианты переходов из одного состояния в другое.

5.2.2 Функциональные диаграммы

Функциональными называют диаграммы, в первую очередь отражающие *взаимосвязи функций* разрабатываемого программного обеспечения.

Отображение взаимосвязи функций активностной модели осуществляется посредством построения *иерархии функциональных диаграмм,* схематически представляющих взаимосвязи нескольких функций. Каждый блок такой диаграммы соответствует некоторой функции, для которой должны быть определены: исходные данные, результаты, управляющая информация и механизмы ее осуществления - человек или технические средства.

Все перечисленные выше связи функции представляются дугами, причем тип связи и ее направление строго регламентированы. Дуги, изображающие каждый тип связей, должны подходить к блоку с определенной стороны, а направление связи должно указываться стрелкой в конце дуги.

Физически дуги исходных данных, результатов и управления представляют собой наборы данных, передаваемые между функциями. Дуги, определяющие механизм выполнения функции - исполнителей-людей или соответствующие технические средства, в основном используют при описании спецификаций сложных информационных систем, которые включают как автоматизированные, так и ручные операции. Блоки и дуги маркируются текстами на естественном языке.

Блоки на диаграмме размещают по «ступенчатой» схеме в соответствии с последовательностью их работы или *доминированием,* которое понимается как влияние, оказываемое одним блоком на другие. В функциональных диаграммах SADT различают пять типов влияний блоков друг на друга:

• вход - выход блока подается на вход блока с меньшим доминированием, т. е. следующего*;*

• управление - выход блока используется как управление для блока с меньшим доминированием (следующего)*;*

• обратная связь по входу - выход блока подается на вход блока с большим

доминированием (предыдущего)*;*

• обратная связь по управлению - выход блока используется как управляющая информация для блока с большим доминированием (предыдущего);

• выход-исполнитель - выход блока используется как механизм для другого блока.

Дуги могут разветвляться и соединяться вместе различными способами. Разветвление означает, что часть или вся информация может использоваться в каждом ответвлении дуги. Дуга всегда помечается до ветвления, чтобы идентифицировать передаваемый набор данных. Если ветвь дуги после ветвления не помечена, то непомеченная ветвь содержит весь набор данных. Каждая метка ветви уточняет, что именно содержит данная ветвь.

Построение иерархии функциональных диаграмм ведется поэтапно с увеличением уровня детализации: диаграммы каждого следующего уровня уточняют структуру родительского блока. Построение модели начинают с единственного блока, для которого определяют исходные данные, результаты, управление и механизмы реализации. Затем он последовательно детализируется с использованием метода пошаговой детализации. При этом рекомендуется каждую функцию представлять не более чем 3-7 блоками. Во всех случаях *каждая подфункция может использовать или продуцировать только те элементы данных, которые использованы или продуцируются родительской функцией,* причем никакие элементы не могут быть опущены, что обеспечивает непротиворечивость построенной модели.

Стрелки, приходящие с родительской диаграммы или уходящие на нее, нумеруют, используя символы и числа. Символ обозначает тип связи: I -входная, С - управляющая, М - механизм, R - результат. Число - номер связи по соответствующей стороне родительского блока, считая сверху вниз и слева направо.

Все диаграммы связывают друг с другом иерархической нумерацией блоков: первый уровень – А0, второй - Al, А2 и т. п., третий — А11, А12, А13 и т. п., где первые цифры - номер родительского блока, а последняя - номер конкретного субблока родительского блока.

Детализацию завершают после получения функций, назначение которых хорошо понятно как заказчику, так и разработчику. Эти функции описывают, используя естественный язык или псевдокоды.

В процессе построения иерархии диаграмм фиксируют всю уточняющую информацию и строят словарь данных, в котором определяют структуры и элементы данных, показанных на диаграммах.

Таким образом, в результате получают спецификацию, которая состоит из иерархии функциональных диаграмм, спецификаций функций нижнего уровня и словаря, имеющих ссылки друг на друга.

5.2.3 Диаграммы потоков данных

Диаграммы потоков данных позволяют специфицировать как функции разрабатываемого программного обеспечения, так и обрабатываемые им данные. При использовании этой модели систему представляют в виде иерархии диаграмм потоков данных, описывающих асинхронный процесс преобразования информации с момента ввода в систему до выдачи пользователю. На каждом следующем уровне иерархии происходит уточнение процессов, пока очередной процесс не будет признан элементарным.

В основе модели лежат понятия внешней сущности, процесса, хранилища (накопителя) данных и потока данных.

Внешняя сущность - материальный объект или физическое лицо, выступающие в качестве источников или приемников информации, например заказчики, персонал, поставщики, клиенты, банк и т. п.

Процесс - преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом. Каждый процесс в системе имеет свой номер и связан с исполнителем, который осуществляет данное преобразование. Как в случае функциональных диаграмм, физически преобразование может осуществляться компьютерами, вручную или специальными устройствами. На верхних уровнях иерархии, когда процессы еще не определены, вместо понятия «процесс» используют понятия «система» и «подсистема», которые обозначают соответственно систему в целом или ее функционально законченную часть.

Хранилище данных - абстрактное устройство для хранения информации. Тип устройства и способы помещения, извлечения и хранения для такого устройства не детализируют. Физически это может быть база данных, файл, таблица в оперативной памяти, картотека на бумаге и т. п.

Поток данных - процесс передачи некоторой информации от источника к приемнику. Физически процесс передачи информации может происходить по кабелям под управлением программы или программной системы или вручную при участии устройств или людей вне проектируемой системы.

Таким образом, диаграмма иллюстрирует как потоки данных, порожденные некоторыми внешними сущностями, трансформируются соответствующими процессами (или подсистемами), сохраняются накопителями данных и передаются другим внешним сущностям - приемникам информации. В результате получают сетевую модель хранения/обработки информации.

Построение иерархии диаграмм потоков данных начинают с диаграммы особого вида - *контекстной диаграммы,* которая определяет наиболее общий вид системы. На такой диаграмме показывают, как разрабатываемая система будет взаимодействовать с приемниками и источниками информации без указания исполнителей, т. е. описывают интерфейс между системой и внешним миром. Обычно начальная контекстная диаграмма имеет форму звезды.

Если проектируемая система содержит большое количество внешних

сущностей (более 10), имеет распределенную природу или включает уже существующие подсистемы, то строят *иерархии* контекстных диаграмм.

При разработке контекстных диаграмм происходит детализация функциональной структуры будущей системы, что особенно важно, если разработка ведется несколькими коллективами разработчиков.

Полученную таким образом модель системы проверяют на полноту исходных данных об объектах системы и изолированность объектов (отсутствие информационных связей с другими объектами).

На следующем этапе каждую подсистему контекстной диаграммы детализируют при помощи диаграмм потоков данных. В процессе детализации соблюдают правило балансировки - *при детализации подсистемы можно использовать компоненты только тех подсистем, с которыми у разрабатываемой подсистемы существует информационная связь* (т. е. с которыми она связана потоками данных).

Решение о завершении детализации процесса принимают в следующих случаях:

- процесс взаимодействует с 2-3 потоками данных;
- возможно описание процесса последовательным алгоритмом;
- процесс выполняет единственную логическую функцию преобразования входной информации в выходную.

На недетализируемые процессы составляют спецификации, которые должны содержать описание логики (функций) данного процесса. Описание может выполняться: на естественном языке, с применением структурированного естественного языка (псевдокодов), с применением таблиц и деревьев решений, в виде схем алгоритмов.

Декомпозицию потоков данных необходимо осуществлять параллельно с декомпозицией процессов.

Окончательно разработку модели выполняют в два этапа.

1этап - *построение контекстной диаграммы* - включает выполнение следующих действий:

- классификацию множества требований и организацию их в основные функциональные группы - процессы;
- идентификацию внешних объектов - внешних сущностей, с которыми система должна быть связана;
- идентификацию основных видов информации - потоков данных, циркулирующей между системой и внешними объектами;
- предварительную разработку контекстной диаграммы;
- изучение предварительной контекстной диаграммы и внесение в нее изменений по результатам ответов на возникающие при изучении вопросы по всем ее частям;
- построение контекстной диаграммы путем объединения всех процессов предварительной диаграммы в один процесс, а также группирования потоков.

2этап - *формирование иерархии диаграмм потоков данных* - включает для каждого уровня:

- проверку и изучение основных требований по диаграмме соответствующего уровня (для первого уровня - по контекстной диаграмме);
- декомпозицию каждого процесса текущей диаграммы потоков данных с помощью детализирующей диаграммы или - если некоторую функцию сложно или невозможно выразить комбинацией процессов, построение спецификации процесса;
- добавление определений новых потоков в словарь данных при каждом появлении их на диаграмме;
- проведение ревизии с целью проверки корректности и улучшения наглядности модели после построения двух-трех уровней.

Полная спецификация процессов включает также описание *структур данных,* используемых как при передаче информации в потоке, так и при хранении в накопителе. Описываемые структуры данных могут содержать альтернативы, условные вхождения и итерации. Условное вхождение означает, что соответствующие элементы данных в структуре могут отсутствовать.

Альтернатива означает, что в структуру может входить один из перечисленных элементов. Итерация означает, что элемент может повторяться некоторое количество раз.

Кроме того, для данных должен быть указан тип: непрерывное или дискретное значение. Для непрерывных данных могут определяться единицы измерений, диапазон значений, точность представления и форма физического кодирования. Для дискретных - может указываться таблица допустимых значений.

Полученную законченную модель необходимо проверить на полноту и согласованность. Под *согласованностью* модели в данном случае понимают выполнение для всех потоков данных *правила сохранения информации:* все поступающие куда-либо данные должны быть считаны и записаны.

6 Проектирование программного обеспечения при структурном подходе

6.1 Разработка структурной и функциональной схем

Процесс проектирования сложного программного обеспечения начинают с определения структурных компонентов и связей между ними. Результат уточнения структуры может быть представлен в виде структурной и/или функциональной схем и описания (спецификаций) компонентов.

Структурная схема - схема, отражающая *состав* и *взаимодействие по управлению* частей разрабатываемого программного обеспечения.

Структурные схемы пакетов и программ не информативны, поскольку организация программ в пакеты не предусматривает передачи управления между ними. Поэтому структурные схемы разрабатывают для каждой программы пакета, а список программ пакета определяют, анализируя функции, указанные в техническом задании.

Самый простой вид программного обеспечения - программа, которая в качестве структурных компонентов может включать только подпрограммы и библиотеки ресурсов. Разработку структурной схемы программы обычно выполняют методом пошаговой детализации.

Структурными компонентами программной системы или программного комплекса могут служить программы, подсистемы, базы данных, библиотеки ресурсов и т. п.

Структурная схема программного комплекса демонстрирует передачу управления от программы-диспетчера соответствующей программе.

Структурная схема программной системы, как правило, показывает наличие подсистем или других структурных компонентов. В отличие от программного комплекса отдельные части (подсистемы) программной системы интенсивно обмениваются данными между собой и, возможно, с основной программой. Структурная же схема программной системы этого обычно не показывает.

Функциональная схема - схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств. Для изображения функциональных схем используют специальные обозначения, установленные стандартом ГОСТ 19.701 90.

Функциональные схемы более информативны, чем структурные. Все компоненты структурных и функциональных схем должны быть описаны. При структурном подходе особенно тщательно необходимо прорабатывать спецификации межпрограммных интерфейсов, так как от качества их описания зависит количество самых дорогостоящих ошибок. К самым дорогим относятся ошибки, обнаруживаемые при комплексном тестировании, так как для их устранения могут потребоваться серьезные изменения уже отлаженных текстов.

6.2 Использование метода пошаговой детализации для проектирования структуры программного обеспечения

Структурный подход к программированию предлагал осуществлять декомпозицию программ методом пошаговой детализации. Результатом декомпозиции является структурная схема программы, которая представляет собой многоуровневую иерархическую схему взаимодействия подпрограмм по управлению.

Метод пошаговой детализации реализует нисходящий подход и базируется на основных конструкциях структурного программирования. Он предполагает

пошаговую разработку алгоритма. Каждый шаг при этом включает разложение функции на подфункции. Так на первом этапе описывают решение поставленной задачи, выделяя общие подзадачи, на следующем аналогично описывают решение подзадач, формулируя при этом подзадачи следующего уровня. Таким образом, на каждом шаге происходит уточнение функций проектируемого программного обеспечения. Процесс продолжают, пока не доходят до подзадач, алгоритмы решения которых очевидны.

Декомпозируя программу методом пошаговой детализации, следует придерживаться основного правила структурной декомпозиции, следующего из принципа вертикального управления: в первую очередь детализировать управляющие процессы декомпозируемого компонента, оставляя уточнение операций с данными напоследок. Это связано с тем, что приоритетная детализация управляющих процессов существенно упрощает структуру компонентов всех уровней иерархии и позволяет не отделять процесс принятия решения от его выполнения: так, определив условие выбора некоторой альтернативы, сразу же вызывают модуль, ее реализующий.

Детализация операций со структурами в последнюю очередь позволит отложить уточнение их спецификаций и обеспечит возможность относительно безболезненной модификации этих структур за счет сокращения количества модулей, зависящих от этих данных.

Кроме этого целесообразно придерживаться следующих рекомендаций:

• не отделять операции инициализации и завершения от соответствующей обработки, так как модули инициализации и завершения имеют плохую связность (временную) и сильное сцепление (по управлению);

• не проектировать слишком специализированных или слишком универсальных модулей, так как проектирование излишне специальных модулей увеличивает их количество, а проектирование излишне универсальных модулей повышает их сложность;

• избегать дублирования действий в различных модулях, так как при их изменении исправления придется вносить во все фрагменты программы, где они выполняются в этом случае целесообразно просто реализовать эти действия в отдельном модуле;

• группировать сообщения об ошибках в один модуль по типу библиотеки ресурсов, тогда будет легче согласовать формулировки, избежать дублирования сообщений, а также перевести сообщения на другой язык.

6.3 Проектирование структур данных

Под проектированием структур данных понимают разработку их представлений в памяти. Основными параметрами, которые необходимо учитывать при проектировании структур данных, являются:

• вид хранимой информации каждого элемента данных;

• связи элементов данных и вложенных структур;

• время хранения данных структуры («время жизни»);

• совокупность операций над элементами данных, вложенными структурами и структурами в целом.

Вид хранимой информации определяет тип соответствующего поля памяти. В качестве элементов данных в зависимости от используемого языка программирования могут рассматриваться:

• целые и вещественные числа различных форматов;

• символы;

• булевские значения: true и false,

а также некоторые структурные типы данных, например:

• строки;

• записи;

• специально объявленные классы.

При этом для числовых полей очень важно правильно определить диапазон возможных значений, а для строковых данных - максимально возможную длину строки.

Связи элементов и вложенных структур, а также их *устойчивость* и *совокупность операций* над элементами и вложенными структурами определяют структуры памяти, используемые для представления данных. *Время жизни* учитывают при размещении данных в статической или динамической памяти, а также во внешней памяти.

Представление данных в оперативной памяти. Различают две базовые структуры организации данных в оперативной памяти: векторную и списковую.

Векторная структура представляет собой последовательность байт памяти, которые используются для размещения полей данных. Последовательное размещение организованных структур данных позволяет осуществлять прямой доступ к элементам: но индексу (совокупности индексов) в массивах или строках или по имени поля в записях или объектах.

Структуры данных в векторном представлении можно размещать как в *статической,* так и в *динамической* памяти. Расположение векторных представлений в динамической памяти иногда позволяет существенно увеличит эффективность использования оперативной памяти. Желательно размещать в динамической памяти временные структуры, хранящие промежуточные результаты, и структуры, размер которых сильно зависит от вводимых исходных; данных.

Списковые структуры строят из специальных элементов, включающих помимо информационной части еще и один или несколько указателей - адресов элементов или вложенных структур, связанных с данным элементом. Размещая подобные элементы в динамической памяти, можно организовывай различные внутренние.

Однако при использовании списковых структур следует помнить, что:

• для хранения указателей необходима дополнительная память;

• поиск информации в линейных списках осуществляется *последовательно,* а потому требует больше времени;

• построение списков и выполнение операций над элементами данных, хранящимися в списках, требует более высокой квалификации программистов, более трудоемко, а соответствующие подпрограммы содержат больше ошибок и, следовательно, требуют более тщательного тестирования.

Обычно векторное представление используют для хранения статических множеств, таблиц (одномерных и многомерных), например матриц, строк, записей, а также графов, представленных матрицей смежности, матрицей инцидентности или аналитически. Списковое представление удобно для хранения динамических (изменяемых) структур и структур со сложными связями.

Представление данных во внешней памяти. Современные операционные системы поддерживают два способа организации данных во внешней памяти: последовательный и с прямым доступом.

При последовательном доступе к данным возможно выполнение только последовательного чтения элементов данных или последовательная их запись.

Прямой доступ возможен только для дисковых файлов, обмен информацией с которыми осуществляется записями фиксированной длины. Адрес записи такого файла можно определить по ее *номеру,* что и позволяет напрямую обращаться к нужной записи.

При выборе типа памяти для размещения структур данных следует иметь в виду, что:

• в оперативной памяти размещают данные, к которым необходим быстрый доступ, как для чтения, так и для их изменения;

• во внешней данные, которые должны сохраняться после завершения программы.

Правильный выбор структур во многом определяет эффективность разрабатываемого программного обеспечения и его технологические качества, поэтому данному вопросу должно уделяться достаточное внимание независимо от используемого подхода.

7 Анализ требований и определение спецификаций программного обеспечения при объектном подходе

На этапе анализа при объектном подходе ставятся две задачи:

•уточнить требуемое поведение разрабатываемого программного обеспечения;

•разработать концептуальную модель его предметной области с точки зрения поставленных задач.

В основе объектного подхода к разработке программного обеспечения лежит *объектная декомпозиция,* т. е. представление разрабатываемого программного обеспечения в виде совокупности объектов, в процессе взаимодействия которых через передачу сообщений и происходит выполнение требуемых функций.

При объектном подходе так же, как при структурном подходе, выполняют декомпозицию программного обеспечения.

UML (Unified Modeling Language унифицированный язык моделирования) - средство описания проектов, создаваемых с использованием объектно-ориентированного подхода.

Спецификация разрабатываемого программного обеспечения при использовании UML объединяет несколько моделей: использования, логическую, реализации, процессов, развертывания.

Модель использования представляет собой описание функциональности программного обеспечения с точки зрения пользователя.

Логическая модель описывает ключевые абстракции программного обеспечения (классы, интерфейсы и т. п.), т. е. средства, обеспечивающие требуемую функциональность.

Модель реализации определяет реальную организацию программных модулей в среде разработки.

Модель процессов отображает организацию вычислений и оперирует понятиями «процессы» и «нити». Она позволяет оценить производительность, масштабируемость и надежность программного обеспечения.

Модель развертывания показывает особенности размещения программных компонентов на конкретном оборудовании.

Таким образом, каждая из указанных моделей характеризует определенный аспект проектируемой системы, а все они вместе составляют относительно полную модель разрабатываемого программного обеспечения.

Всего UML предлагает девять дополняющих друг друга диаграмм, входящих в различные модели:

- диаграммы вариантов использования;
- диаграммы классов;
- диаграммы пакетов;
- диаграммы последовательностей действий;
- диаграммы кооперации;
- диаграммы деятельностей;
- диаграммы состояний объектов;
- диаграммы компонентов;
- диаграммы размещения.

Диаграммы используют единую графическую нотацию.

Помимо указанных диаграмм, как и при структурном подходе, спецификация включает словарь терминов, описания и текстовые спецификации.

Конкретный набор документации определяется разработчиком.

7.1 Определение «вариантов использования»

Разработку спецификаций программного обеспечения начинают с анализа требований к функциональности, указанных в техническом задании. В процессе анализа выявляют внешних пользователей разрабатываемого программного обеспечения и перечень отдельных аспектов поведения в процессе взаимодействия с конкретными пользователями. Аспекты поведения программного обеспечения были названы «вариантами использования» или «прецедентами» (use cases).

Вариант использования представляет собой характерную процедуру применения разрабатываемой системы конкретным действующим лицом, в качестве которого могут выступать не только люди, но и другие системы или устройства.

В зависимости от цели выполнения конкретной процедуры различают следующие варианты использования:

- основные обеспечивают требуемую функциональность разрабатываемого программного обеспечения;
- вспомогательные - обеспечивают выполнение необходимых настроек: системы и ее обслуживание (например, архивирование информации и т. п.);
- дополнительные обеспечивают дополнительные удобства для пользователя (как правило, реализуются в том случае, если не требуют серьезных затрат каких-либо ресурсов ни при разработке, ни при эксплуатации).

Вариант использования можно описать кратко или подробно. Краткая форма описания содержит: название варианта использования, его цель, действующих лиц, тип варианта использования (основная, второстепенная или дополнительная) и его краткое описание.

Основные варианты использования обычно описывают подробно, стараясь отразить особенности предметной области разрабатываемого программного обеспечения. Подробная форма, кроме указанной выше информации, включает описание типичного хода событий и возможных альтернатив. Типичный ход событий представляют в виде диалога между пользователями и системой, последовательно нумеруя события. Если пользователь может выбирать варианты, то их описывают в отдельных таблицах. Также отдельно приводят альтернативы, связанные с нарушением типичного хода событий.

7.1.1 Диаграммы вариантов использования

Диаграммы вариантов использования позволяют наглядно представить ожидаемое поведение системы. Основными понятиями диаграмм вариантов использования являются: действующее лицо, вариант использования, связь.

Действующее лицо внешняя по отношению к разрабатываемому программному обеспечению сущность, которая взаимодействует с ним с целью

получения или предоставления какой-либо информации. Как уже упоминалось выше, действующими лицами могут быть пользователи, другое программное обеспечение или какие-либо технические средства, взаимодействующие с разрабатываемым программным обеспечением.

Вариант использования некоторая очевидная для действующего лица процедура, решающая его конкретную задачу. Все варианты использования, так или иначе, связаны с требованиями к функциональности разрабатываемой системы и могут сильно отличаться по объему выполняемой работы.

Связь взаимодействие действующих лиц и соответствующих вариантов использования.

Варианты использования также могут быть связаны между собой.

Использование подразумевает, что существует некоторый фрагмент поведения разрабатываемого программного обеспечения, который повторяется в нескольких вариантах использования. Этот фрагмент оформляют как отдельный вариант использования и указывают связь с ним типа «использование».

Расширение применяют, если имеется два подобных варианта использования, различающиеся наличием в одном из них некоторых дополнительных действий. В этом случае дополнительные действия определяют как отдельный вариант использования, который связан с основным вариантом связью типа «расширение».

7.2 Построение концептуальной модели предметной области

7.2.1 Диаграммы классов

Диаграммы классов — центральное звено объектно-ориентированных методов разработки программного обеспечения. UML предлагает использовать три уровня диаграмм классов в зависимости от степени их детализации:

• концептуальный уровень, на котором диаграммы классов, называемые в этом случае контекстными, демонстрируют связи между основными понятиями предметной области;

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

• уровень реализации, на котором диаграммы классов непосредственно показывают поля и операции конкретных классов.

Каждую из перечисленных моделей используют на конкретном этапе разработки программного обеспечения:

• концептуальную модель - на этапе анализа;

• диаграммы классов уровня спецификации - на этапе проектирования;

• диаграммы классов уровня реализации - на этапе реализации.

Концептуальные модели в соответствии с определением оперируют понятиями предметной области, атрибутами этих понятий и отношениями между ними. Понятию в предметной области разрабатываемого программного

обеспечения могут соответствовать как материальные предметы, так и абстракции, которые применяют специалисты предметной области.

Основным понятиям в модели ставятся в соответствие классы. *Класс* при этом традиционно понимают как совокупность общих признаков заданной группы объектов предметной области. В соответствии с этим определением на диаграмме классов каждому классу соответствует группа объектов, общие признаки которых и фиксирует класс. Так класс Студент объединяет общие признаки группы людей, обучающихся в высших учебных заведениях. Экземпляр класса или объект (например, И.И. Иванов) обязательно обладает всей совокупностью признаков своего класса и может иметь собственные признаки, не фиксированные в классе. Так, например, помимо того, что И.И. Иванов является студентом, он еще может быть спортсменом, музыкантом и т. д. Строго говоря, таким собственным признаком является и идентифицирующее студента имя.

На диаграммах класс изображается в виде прямоугольника, внутри которого указано имя класса. При необходимости допускается указывать характеристики класса, например атрибуты, используя специальные секции условного обозначения.

7.2.2 Диаграмма последовательностей системы

Системные события и операции. *Диаграмма последовательностей системы* - графическая модель, которая для определенного сценария варианта использования показывает генерируемые действующими лицами события и их порядок. При этом система рассматривается как единое целое.

Для построения диаграммы последовательностей системы необходимо:

− представить систему как «черный ящик» и изобразить для нее *линию жизни* - вертикальную пунктирную линию, подходящую к блоку снизу;

− идентифицировать каждое действующее лицо и изобразить для него линию жизни (много действующих лиц бывает в вариантах совместного использования программного обеспечения);

− из описания варианта использования определить множество системных событий и их последовательность;

− изобразить системные события в виде линий со стрелкой на конце между линиями жизни действующих лиц и системы, а также указать имена событий и списки передаваемых значений.

В отличие от внутренних, события, которые генерируются для системы действующими лицами, называют *системными.* Системные события инициируют выполнение соответствующего множества операций, также называемых *системными.* Каждую системную операцию называют по имени соответствующего сообщения.

Множество всех системных операций определяют, идентифицируя системные события всех вариантов использования. Для наглядности системны операции изображают в виде операций абстрактного класса (типа) System. Если

необходимо разделить множество операций на подмножества, инициируемые разными пользователями, то используют несколько абстрактных классов: System1, System2 и т. д.

Каждую системную операцию необходимо описать. Обычно описание системной операции содержит:

- имя операции и ее параметры;
- описание обязанности;
- указание типа;
- названия вариантов использования, в которых она используется;
- примечания для разработчиков алгоритмов и т. д.;
- описание обработки возможных исключений;
- описание вывода неинтерфейсных сообщений;
- предположение о состоянии системы до выполнения операции (предусловие);
- описание изменения состояния системы после выполнения операции (постусловие).

7.2.3 Диаграммы деятельностей

В зависимости от степени детализации диаграммы деятельностей так же, как диаграммы классов, используют на разных этапах разработки. На этапе анализа требований и уточнения спецификаций диаграммы деятельностей позволяют конкретизировать основные функции разрабатываемого программного обеспечения.

Под *деятельностью* в данном случае понимают задачу (операцию), которую необходимо выполнить вручную или с помощью средств автоматизации. Каждому варианту использования соответствует своя последовательность задач. В теоретическом плане диаграммы деятельности являются обобщенным представлением алгоритма, реализующего анализируемый вариант использования. На диаграмме деятельность обозначается прямоугольником с закругленными углами.

Диаграммы деятельностей позволяют описывать альтернативные и параллельные процессы. Для обозначения альтернативных процессов используют ромб, условие указывают над ним слева или справа, а альтернативы «да», «нет» - рядом с соответствующими выходами. С помощью этого же блока можно построить циклический процесс. Множественность активации деятельности обозначают символом «*», помещенным рядом со стрелкой активации деятельности, и при необходимости уточняют надписью вида «для каждой строки».

Для обозначения параллельных процессов используют линейки синхронизации, причем условие синхронизации можно уточнить, указав его на диаграмме.

На этапе определения спецификаций имеет смысл уточнять только варианты использования, краткое описание которых недостаточно для

понимания сущности решаемых проблем. Диаграммы деятельностей, таким образом, можно использовать вместо описания вариантов использования или как дополнение к ним.

8 Проектирование программного обеспечения при объектном подходе

Основной задачей *логического проектирования* при объектном подходе является разработка классов для реализации объектов, полученных при объектной декомпозиции, что предполагает полное описание полей и методов каждого класса.

Физическое проектирование при объектном подходе включает объединение классов и других программных ресурсов в программные компоненты, а также размещение их компонентов на конкретных вычислительных устройствах.

8.1 Разработка структуры программного обеспечения при объектном подходе

Большинство классов можно отнести к определенному типу, который применительно к данному подходу называют стереотипом, например:

- классы-сущности (классы предметной области);
- граничные (интерфейсные) классы;
- управляющие классы;
- исключения и т. д.

8.1.1 Диаграмма пакетов

Диаграмма пакетов показывает, из каких частей состоит проектируемая программная система, и как эти части связаны друг с другом.

Связь между пакетами фиксируют, если изменения в одном пакете могут повлечь за собой изменения в другом. Она определяется внешними связями классов и других ресурсов, объединенных в пакет. Возможны различные виды зависимости классов, например:

- объекты одного класса посылают сообщения объектам другого класса;
- объекты одного класса обращаются к компонентам объектов другого;
- объекты одного класса используют объекты другого в списке параметров методов и т. п.

Самыми хорошими технологическими характеристиками отличается вариант, при котором каждый пакет включает интерфейс, содержащий описание всех ресурсов данного пакета, и взаимодействие пакетов осуществляется только через этот интерфейс. Изменения реализации ресурсов пакета в этом случае не затрагивает других пакетов. И только изменения в интерфейсе могут потребовать изменения пакетов, использующих ресурсы данного пакета.

Пакеты, с которыми связаны все пакеты программной системы, называют

глобальными. Интерфейсы таких пакетов необходимо проектировать особенно тщательно, так как изменения в них потребуют проверки всех пакетов разрабатываемой системы.

8.1.1.1 Определение отношений между объектами

После определения основных пакетов разрабатываемого программного обеспечения переходят к детальному проектированию классов, входящих в каждый пакет. Классы-кандидаты, которые предположительно должны войти в конкретный пакет, показывают на диаграмме классов этапа проектирования и уточняют отношения между объектами указанных классов.

8.1.2 Диаграммы последовательностей этапа проектирования

Диаграммы последовательностей этапа проектирования отображают взаимодействие объектов, упорядоченное по времени. В отличие от диаграмм последовательности этапа анализа на ней показывают внутренние объекты, а также последовательность сообщений, которыми обмениваются объекты в процессе реализации фрагмента варианта использования, называемого сценарием.

Объекты изображают в виде прямоугольников, внутри которого указана информация, идентифицирующая объект: имя, имя объекта и имя класса или только имя класса.

Каждое сообщение представляют в виде линии со стрелкой, соединяющей линии жизни двух объектов. Эти линии помещают на диаграмму в порядке генерации сообщений (сверху вниз и слева направо). Сообщению присваивают имя, но можно указать и аргументы, и управляющую информацию, например условие формирования или маркер итерации (*). Возврат при передаче синхронных сообщений подразумевают по умолчанию.

Если объект создается сообщением, то его рисуют справа от стрелки сообщения так, чтобы стрелка сообщения входила в него слева.

Диаграммы последовательностей также позволяют изображать параллельные процессы. Асинхронные сообщения, которые не блокируют работу вызывающего объекта, показывают половинкой стрелки. Такие сообщения могут:

- создавать новую ветвь процесса;
- создавать новый объект;
- устанавливать связь с уже выполняющейся ветвью процесса.

На линии жизни в этом случае дополнительно показывают активации, которые обозначаются прямоугольником, наложенным поверх линии жизни.

Уничтожение объекта показывают большим знаком «X».

При необходимости линию жизни можно прервать, чтобы не уточнят обработку, не связанную с анализируемыми объектами.

8.1.3 Диаграмма кооперации

Диаграмма кооперации - это альтернативный способ представления взаимодействия объектов в процессе реализации сценария, который позволяет

по-другому взглянуть на ту же информацию. В отличие от диаграмм последовательностей диаграммы кооперации показывают потоки данных между объектами классов, что позволяет уточнить связи между ними.

8.1.3.1 Уточнение отношений классов

Процесс проектирования классов начинают с уточнения отношений между ними. На этапе проектирования, помимо ассоциации и обобщения, различают еще два типа отношения между классами: агрегацию и композицию.

Диаграммы классов позволяют также отобразить ограничения, которые невозможно показать, используя только понятия, рассмотренные выше (ассоциации, обобщения, атрибуты, операции). Подобную информацию на диаграмме классов можно представить в виде записи на естественном языке или в виде математической формулы, поместив их в фигурные скобки.

8.1.3.2 Интерфейсы

Интерфейсом в UML называют класс, содержащий только объявление операций. Отдельное описание интерфейсов улучшает технологические качества проектируемого программного обеспечения. Интерфейсы широко применяют при разработке сетевого программного обеспечения, которое должно идентично функционировать в гетерогенных средах, а также для организации взаимодействия с системами управления базами данных и т. п., так как механизм полиморфного наследования позволяет создавать различные реализации одного и того же интерфейса.

С точки зрения теории объектно-ориентированного программирования интерфейс представляет собой особый вид абстрактного класса, отличающийся тем, что он не содержит методов, реализующих указанные операции, и объявлений полей. Другими словами, абстрактные классы позволяют определить реализацию некоторых методов, а интерфейсы требуют отложить определение всех методов.

На диаграмме классов интерфейс можно показать двумя способами: с помощью специального условного обозначения или объявив для класса стереотип «Interface».

Для остальных классов, ассоциированных с интерфейсом, следует уточнить ассоциацию, показав отношение зависимости. Это отношение в данном случае означает, что класс использует указанный интерфейс, т. е. класса от интерфейса обращается к описанным в интерфейсе функциям.

На диаграмме классов целесообразно также указать множественность объектов. Поскольку каждый раз решается одна задача с единственными данными, используя конкретный алгоритм, и в результате получают единственное решение, все перечисленные выше ассоциации связывают объекты «один к одному».

8.2 Проектирование классов

Собственно проектирование классов предполагает окончательное определение структуры и поведения его объектов. Структура объектов определяется совокупностью атрибутов и операций класса. Каждый атрибут - это поле или совокупность полей данных, содержащихся в объекте класса.

Поведение объектов класса определяется реализуемыми обязанностями. Обязанности выполняются посредством операций класса.

Таким образом, при проектировании класса помимо имени и максимально полного списка атрибутов необходимо уточнить его ответственность и операции. Причем как атрибуты, гак и операции в процессе проектирования целесообразно дополнительно специфицировать. В зависимости от степени детализации диаграммы классов обозначение атрибута может помимо имени включать: тип, описание видимости и значение по умолчанию.

Исходный список операций класса формируют, анализируя диаграммы деятельностей, диаграммы взаимодействия и диаграммы последовательностей действий, построенные для различных сценариев с участием объектов проектируемого класса. На начальных этапах проектирования в секции операций класса обычно указывают лишь имена основных операций, определяющих наиболее общее поведение объектов соответствующих классов. Но мере уточнения добавляют новые операции, а информацию об уже имеющихся операциях детализируют.

Большинство атрибутов выявляется при анализе предметной области, требований технического задания и описаний потоков событий.

Если объекты проектируемого класса должны реализовывать сложное поведение, для них целесообразно разрабатывать диаграммы состояний.

8.2.1 Диаграммы состояний объекта.

Под состоянием объекта применительно к диаграмме состояний понимают ситуацию в жизненном цикле объекта, во время которой он: удовлетворяет некоторому условию, осуществляет определенную деятельность или ожидает некоторого события. Изменение состояния, связанное с нарушением условия или соответственно завершением деятельности или наступлением события, называют переходом.

Диаграммы состояний показывают состояния объекта, возможные переходы, а также события или сообщения, вызывающие каждый переход.

Условие записывается в виде логического выражения. Переход происходит; если результат выражения «истина». Объект не может одновременно перейти в два разных состояния, поэтому условия перехода для любого события должны быть взаимоисключающими.

В отличие от деятельностей, действия, указанные для перехода, связывают с последним и рассматривают как мгновенные и непрерываемые.

При необходимости можно определять суперсостояния, которые объединяют несколько состояний в одно. Этот механизм обычно используют, чтобы показать переход из нескольких состояний в одно и то же состояние,

например, при отмене каких-либо действий.

8.2.2 Проектирование методов класса.

Достаточно существенную информацию о действиях, которые должны выполняться методами класса, можно получить, анализируя диаграммы последовательности действий. Однако алгоритмы всех сколько-нибудь сложных методов необходимо проработать детально. При этом можно использовать как уже известные нотации (схемы алгоритмов и псевдокоды), так и диаграммы деятельностей.

Диаграммы деятельностей могут использоваться и при проектировании методов обработки сообщений, в том числе и затрагивающих несколько объектов. В последнем случае целесообразно указать вертикальными пунктирными линиями ответственности объектов соответствующих классов, что позволит проследить вызовы других объектов.

Следует помнить, что в соответствии с общими правилами процедурной декомпозиции любую деятельность можно декомпозировать и изобразить в виде диаграммы деятельности более низкого уровня.

8.2.3 Компоновка программных компонентов.

Диаграммы компонентов применяют при проектировании физической структуры разрабатываемого программного обеспечения. Эти диаграммы показывают, как выглядит программное обеспечение на физическом уровне, т. е. из каких частей оно состоит и как эти части связаны между собой.

Диаграммы компонентов оперируют понятиями компонент и зависимость. Под компонентами при этом понимают физические заменяемые части программного обеспечения, которые соответствуют некоторому набору интерфейсов и обеспечивают их реализацию.

Зависимость между компонентами фиксируют, если один компонент содержит некоторый ресурс (модуль, объект, класс и т. д.), а другой его использует. Качество компоновки оценивают по количеству и типу связей между компонентами, т. е. по степени независимости компонентов. На диаграмме компонентов зависимость обозначают пунктиром со стрелкой на конце.

Кроме этого на диаграмме компонентов допустимо уточнять зависимость между компонентами, используя обозначения обобщения, ассоциации, композиции, агрегатирования и реализации.

При «сборке» исполняемых файлов диаграммы компонентов применяют для отображения взаимосвязей файлов, содержащих исходный код.

Для программного обеспечения с архитектурой «клиент-сервер», диаграмму компонентов можно использовать в качестве структурной схемы, определяющей архитектуру разрабатываемого программного обеспечения, так как она позволяет показать связи по управлению частей системы (компонентов). Однако при проектировании закую схему необходимо уточнить, показав более подробно состав компонентов разрабатываемой системы.

8.3 Проектирование размещения программных компонентов для распределенных программных систем

При физическом проектировании распределенных программных систем необходимо определить наиболее оптимальный вариант размещения программных компонентов на реальном оборудовании в локальной или глобальной сетях. Для этого используют специальную модель UML-диаграмму размещения.

Диаграмма размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Каждой части аппаратных средств системы, например компьютеру или датчику, на диаграмме размещения соответствует узел. Соединения узлов означают наличие в системе соответствующих коммуникационных каналов. Внутри узлов указывают размещенные на данном оборудовании программные компоненты разрабатываемой программной системы, сохраняя указанные на диаграмме компонентов отношения зависимости.

С точки зрения диаграммы размещения локальная и глобальная сети - это тоже узлы, которые обладают некоторой спецификой.

9 Требования к оформлению отчета

Рекомендуется в тексте пояснительной записки использовать графики, схемы, диаграммы и другие иллюстрационные материалы, наглядно представляющие процесс и результаты проектирования.

Общий объем записки к курсовому проекту – 30-40 страниц формата А4. Материал излагается по разделам в соответствии с содержанием, над каждым разделом дается соответствующее название.

Необходимо стремиться к ясности и самостоятельности изложения, не повторять текстов из литературных источников. Все цитаты, заимствованные цифры и факты должны иметь ссылки на источники.

Все материалы сшиваются в папку. Материал проекта располагается в следующем порядке:

1. Титульный лист (Приложение А)
2. Задание на проектирование (Приложение Б)
3. Содержание (Приложение В)
4. Скомплектованная по разделам текстовая часть с иллюстрациями
5. Заключение
6. Список использованных источников
7. Приложения
8. Диск с программой в отдельном конверте.

При защите курсового проекта следует иметь при себе реализованные материалы на машинных носителях, презентацию.

9.1 Общие требования

1) Пояснительная записка курсового проекта выполняется в печатном виде. Текст располагается на бумаге формата А4 (210×297 мм).

2) Повреждение листов, помарки текста или графики не допускаются.

3) Листы текстового документа должны быть сброшюрованы.

4) Параметры страницы: слева – 2,0; справа – 1,0; сверху и снизу – 2,0.

5) Абзацы в тексте начинают отступом 15-17 мм.

6) Размер шрифта Times New Roman должен быть 14 пунктов, расстояние между строками один интервал.

7) Текст документа при необходимости разделяют на разделы и подразделы. Разделы, подразделы должны иметь заголовки. Заголовки следует печатать с прописной буквы без точки в конце, не подчеркивая. Переносы слов в заголовках не допускаются. Точку в конце заголовка не ставят. Расстояние между заголовком и последующим текстом должно быть 30 пунктов (15 мм). Расстояние между заголовками раздела и подраздела - 18 пунктов (8 мм). Расстояние между последней строкой текста и последующим заголовком - 30 пунктов (15 мм).

8) Каждый раздел рекомендуется начинать с новой страницы. Разделы пояснительной записки должны иметь порядковые номера, обозначенные арабскими цифрами без точки, в пределах всей пояснительной записки и записанные с абзацного отступа.

9) Подразделы следует нумеровать арабскими цифрами в пределах каждого раздела. Номер подраздела состоит из номера раздела и подраздела, разделенных точкой. В конце номера подраздела точку не ставят.

10) Нумерация страниц пояснительной записки должна быть сквозной. Первой страницей является титульный лист. На титульном листе номер не ставят.

11) Приложения не нумеруются.

12) На первом (заглавном) листе помещают содержание, включающее номера и наименования разделов и подразделов с указанием номеров листов. Содержание включают в общее количество листов документа. Слово "Содержание" записывается в виде заголовка (симметрично тексту) с прописной буквы. Наименования, включенные в содержание, записывают строчными буквами, начиная с прописной буквы.

9.2 Изложение текста пояснительной записки

1) Текст пояснительной записки должен быть кратким, четким и не

допускать различных толкований.

2) В тексте пояснительной записки не допускается:

—применять сокращения слов, кроме установленных правилами и соответствующими государственники стандартами;

—сокращения типа: шт.; экз.; разд.; п.; рис.; поз.; табл. следует применять только в сопровождении цифр. Например: 10 экз.; п.4; разд. 2.;

—использовать в тексте математический знак минус (-) перед отрицательными значениями величин. Следует писать слово "минус":

—употреблять без цифр математические знаки (=<, => и т.п.), а также знаки № (номер), % (процент) и т.д.

3) Допускается применять аббревиатуру (сокращения терминов, состоящих из нескольких слов) и сокращенные наименования изделий. Но при первом упоминании обязательно приводится их полное название и в скобках сокращенное. Например: АСУ - автоматизированная система управления.

4) Наименования команд, режимов, сигналов и т.п. в тексте следует выделять кавычками.

5) Иллюстрации.

—Все иллюстрации (диаграммы, схемы, чертежи) именуются рисунками. Рисунки, если их в тексте больше одного, нумеруются последовательно арабскими цифрами сквозной нумерацией, например: (Рисунок 3). Допускается нумерация иллюстраций в пределах раздела (Рисунок 2.3).

—Рисунки могут иметь наименования и поясняющие данные (подрисуночный текст). Слово "Рисунок" и наименование рисунка помещают после пояснительных данных и располагают следующим образом: Рисунок 3 - Медведь.

При ссылках на иллюстрации следует писать "... в соответствии с рисунком 3".

6) Таблицы.

—Цифровой материал, как правило, оформляют в виде таблиц. Заголовки граф таблиц начинают с прописных букв, а подзаголовки - со строчных, если они составляют одно предложение с заголовком. Если подзаголовки имеют самостоятельные значения, то их начинают с прописных букв. Заголовки и подзаголовки указывают в единственном числе.

—Если строки или графы выходят за формат листа, таблицу делят на части, которые в зависимости от таблицы переносят на другие листы, или помещают на одном листе рядом, или одна под другой.

—Слово "Таблица", заголовок (при его наличии) и порядковый номер таблицы помещают один раз слева над первой частью таблицы. Например:

Таблица 15 - Сравнительные характеристики элементов

Над последующими частями таблицы помещают слова "Продолжение таблицы " с указанием номера. Если в конце страницы таблица прерывается, и ее продолжение будет на следующей странице, то в первой части таблицы нижнюю

горизонтальную линию, ограничивающую таблицу, не проводят.

—Таблицы, за исключением таблиц приложений, следует нумеровать арабскими цифрами сквозной нумерацией. Допускается нумерация таблиц в пределах раздела. В этом случае номер таблицы состоит из номера раздела и порядкового номера таблицы, разделенных точкой.

—При ссылке на таблицу указывают слово "таблица" и ее номер.

7) Формулы.

—В формулах в качестве символов следует применять обозначения, установленные соответствующими стандартами. Перенос формул допускается только на знаках +, -, *, =, причем на новой строке знак необходимо повторить. Расшифровку символов с указанием единиц физических величин и числовых коэффициентов, входящих в формулу, дают с новой строки в той же последовательности, в какой они приведены в формуле. Первая строка расшифровки должна начинаться со слова "где" без двоеточия после него, например:

$$I = U / R, \qquad (21)$$

где I - сила тока, А;

—Формулы следует выделять из текста в отдельную строку. Выше и ниже каждой формулы должно быть оставлено не менее одной свободной строки.

—Формулы должны нумероваться сквозной нумерацией арабскими цифрами. Номер указывают справа на уровне формулы в круглых скобках, например:

$$V = M / W, \qquad (12)$$

—Ссылки в тексте на номер формулы дают в скобках, например: "... в формуле (12)". Допускается нумерация формул в пределах раздела.

8) Список использованной литературы и приложения.

—В список использованной литературы включаются все источники, которые следует располагать в порядке появления ссылок в тексте.

—Сведения о книгах (монографии, учебники, справочники и т. д.) должны включать: фамилию и инициалы автора, заглавие книги, место издания, издательство, год издания, количество страниц в книге. Допускается сокращение названия трех городов - Москва (М.).Санкт-Петербург (С.-Пб.), Киев (К.). Например:

Шляндин В.И. Цифровые измерительные устройства. - М.: Высшая школа. 1991. - 335 с.

—Иллюстрационный материал, таблицы или тексты вспомогательного характера допускается давать в виде приложений. Приложения оформляются как продолжение пояснительной записки. Каждое приложение должно начинаться с нового листа с указанием наверху посередине страницы слова "Приложение". Приложение должно иметь тематический заголовок, который записывают симметрично текста с прописной буквы отдельной строкой.

—Приложения обозначают заглавными буквами русского алфавита, начиная с А, за исключением букв Е, З, Й, О, Ч, Ь, Ы, Ъ. Если в документе одно приложение, оно обозначается "Приложение А".

Приложение А
Пример оформления титульного листа

Министерство образования Иркутской области
Государственное бюджетное профессиональное
образовательное учреждение Иркутской области
«Иркутский авиационный техникум»
(ГБПОУИО «ИАТ»)

КП.09.02.03.16.09.01.ПЗ

ИНФОРМАЦИОННАЯ СИСТЕМА «ПОРТФОЛИО СТУДЕНТА»

Председатель ВЦК: _____ (М.А. Богачева)
(подпись, дата)

Руководитель: _____ (М.А. Кудрявцева)
(подпись, дата)

Студент: _____ (Р.А. Абрамов)
(подпись, дата)

Иркутск 2016

Приложение Б

Задание на курсовой проект

Министерство образования Иркутской области
Государственное бюджетное профессиональное образовательное учреждение
Иркутской области
«Иркутский авиационный техникум»
(ГБПОУИО «ИАТ»)

УТВЕРЖДАЮ:
Председатель ВЦК
_____/М.А. Богачева/
22 сентября 2016 г.

ЗАДАНИЕ
на курсовой проект

по МДК.03.01. «Технология разработки программного обеспечения»
студенту IV курса учебной группы ПКС-9
Абрамову Ростиславу Алексеевичу
(фамилия, имя, отчество)

Тема: Информационная система «Портфолио студента»

Начало проектирования: 22 сентября 2016 г.
Срок представления к защите: 12 декабря 2016 г.

Руководитель: _____ (М.А. Кудрявцева)
(подпись, дата)

Студент: _____ (Р.А. Абрамов)
(подпись, дата)

Продолжение приложения Б

Задание:

Разработать информационную систему в соответствии с предметной областью.

Требования к работе информационной системы

- Система должна быть разработана в соответствии техническим заданием.
- Система должна оперативно работать с данными в данной предметной области.
- Сведения должны храниться и использоваться при составлении и отборе статистических данных.
- На этапе проектирования необходимо провести моделирование (составить модели), описывающее процесс создания и эксплуатации автоматизированной информационной системы.
- Работа с исходными данными должна предусматривать режим коллективного доступа.
- Пользователь должен иметь возможность влиять на результаты расчетов путем выбора параметров данных, вида требуемой операции.
- Работа пользователя с системой должна строиться в виде диалогового процесса с последовательным выбором действий.
- Должна быть предоставлена возможность протоколирования основных результатов работы с выводом в файл или на печать.
- Результатами работы информационной системы является выполнение следующих операций:
- изменение хранимых данных (ввод, редактирование, удаление данных), обеспечение целостности данных;
- поиск или отбор данных и их представление в соответствии с условиями, сформулированными «заказчиком»;
- преобразование хранимых данных и/или формирование новых данных в результате выполнения процедур обработки;
- документирование результатов обработки данных;
- руководство пользователя в составе программного комплекса (Help).

В результате выполнения курсового проекта должен быть выполнен полный цикл проектирования и разработки информационной системы, включая информационное, программное и документальное обеспечение.

Методические указания по выполнению программной документации

Текст пояснительной записки оформляется в соответствии с ГОСТ 7.32-2001. Страницы текста и включенные в ПЗ иллюстрации и таблицы должны соответствовать формату А4 по ГОСТ 9327. Программная документация, входящая в состав курсового проекта должна соответствовать требованиям ЕСПД.

Материалы, представляемые к защите

Пояснительная записка. Программная часть. Презентация.

CD с материалами курсового проекта.

Приложение В

Пример оформления содержания

Содержание