ЮГОРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Институт прикладной математики, информатики и управления
Кафедра автоматизированных систем обработки информации и управления

ПРОЕКТИРОВАНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

Ханты-Мансийск
2010

ББК 32.973.26-02
УДК 004.415.2
П-79

Рецензент:
доктор физ.-мат. наук, профессор, заведующий кафедрой высшей математики
Пятков С.Г.

П-79 Проектирование автоматизированных систем обработки информации и управления: Методические указания по курсовому проектированию / Сост. Мелихов А.Ю. – Ханты-Мансийск: Информационно-издательский центр ЮГУ, 2010. – 87 с.

Методические указания служат руководством для выполнения курсового проекта по дисциплине “Проектирование автоматизированных систем обработки информации и управления” для студентов высшего профессионального образования очной формы обучения направлений 230100 – Информатика и вычислительная техника, 010400 – Прикладная математика и информатика и 010200 – Математика и компьютерные науки.

ББК 32.973.26-02
УДК 004.415.2

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Автоматизированной системой обработки информации и управления (АСОИУ) называют систему, состоящую из взаимосвязанных и взаимодействующих в пространстве и во времени вычислительных, алгоритмических средств, а также средств связи, источников информации с метрологическим обеспечением, средств управления и отображения, применяемых для получения продукта заданного качества в определенных условиях эксплуатации с участием человека. Степень автоматизации функций обработки и управления определяется производственной необходимостью, возможностями формализации автоматизируемого технологического процесса и **должна быть экономически (и) или социально обоснована** [66].

Выполнение курсового проекта (КП) по дисциплине «Проектирование автоматизированных систем обработки информации и управления» («Проектирование информационно-моделирующих систем (ИМС)» - специальность 010501) предусматривает разработку **проекта по созданию новой** или **развитию существующей системы**.

Процесс создания АСОИУ представляет собой комплекс научно- исследовательских, предпроектных, проектных, строительных, монтажно-наладочных работ, испытаний, опытную эксплуатацию АСОИУ, а также подготовку и обучение персонала и работы по подготовке объекта управления к вводу АСОИУ в эксплуатацию. При создании АСОИУ необходимо, с одной стороны, руководствоваться принципами системности, развития, совместимости, эффективности, стандартизации и унификации, с другой стороны, необходимо максимально использовать типовые проектные решения, пакеты прикладных программ, унифицированные проекты, применять для новых объектов управления ранее созданные проекты АСОИУ [66, 67]. При создании, функционировании и развитии АСОИУ необходимо оценивать научно-технический уровень системы с целью проверки его соответствия последним достижениям науки и техники.

Развитие (модификация) АСОИУ заключается в пополнении и (или) обновлении функций и видов ее обеспечения путем доработки программных и (или) технических средств или настройки имеющихся средств [66].

Как в случае создания, так и в случае развития существующей системы **должны преследоваться следующие цели** [75:10]:

- экономия топлива, сырья, материалов и др. производственных ресурсов;
- обеспечение или повышение надежности (безопасности) функционирования объекта или протекания технологического процесса;
- повышение качества выходного продукта;
- снижение затрат труда;
- достижение оптимальной загрузки оборудования;
- оптимизация режимов работы технологического оборудования и т.п.

Выполнение курсового проекта является завершающим этапом освоения дисциплины «Проектирование АСОИУ» («Проектирование ИМС») и представляет собой самостоятельную разработку студентами проектной документации по созданию АСОИУ до стадии «4. Эскизный проект» включительно по ГОСТ 34.601-90 [70].

Настоящие методические указания содержат рекомендации по выполнению и защите курсового проекта, а также требования, предъявляемые к содержанию и оформлению пояснительной записки (ПЗ).

Для облегчения работы с методическими указаниями в тексте использовались специальные обозначения:

- знаком «  » отмечены положения или определения, требующие особого внимания при проработке курсового проекта;
- знаком  отмечены вопросы, уровню проработки которых будет уделяться особое внимание при проверке пояснительной записки.

1. ЦЕЛИ КУРСОВОГО ПРОЕКТИРОВАНИЯ

Самостоятельная разработка проектной документации подготавливает студента к успешному выполнению выпускной квалификационной работы и является важным этапом в формировании специалиста в области автоматизации процессов обработки информации и управления технологическими процессами в заданной предметной области.

Работа над курсовым проектом требует решения следующих **задач**:

- ознакомление с основными этапами проектирования;
- приобретение навыков исследования процессов обработки информации, моделирования и управления в заданной предметной области;
- изучение нормативной документации в области проектирования и разработки АСОИУ;
- изучение инструментальных средств разработки АСОИУ на базе CASE-технологий;
- приобретение навыков анализа и составления технического задания (ТЗ);
- выработка технически обоснованных решений;
- разработка проекта системы, отвечающей требованиям заказчика;
- закрепление навыка поиска научно - технической литературы и работы с ней;
- освоение правил оформления проектной документации.

В процессе проектирования студент решает творческие задачи и оформляет конструкторские документы, которые присущи стадиям формирования требований, разработки концепции, разработки технического задания и эскизного проекта (в соответствии с ГОСТ 34.601-90 [70]).

2. ТЕМАТИКА КУРСОВЫХ ПРОЕКТОВ

Содержание курсового проекта (КП) состоит в проектировании информационной системы (подсистемы), обеспечивающей решение одной или нескольких задач автоматизации обработки информации, моделирования и/или управления в заданной предметной области с использованием современных средств вычислительной техники и средств телекоммуникаций, а также современных информационных технологий.

При выборе темы КП необходимо руководствоваться следующими требованиями: актуальность; практическая ценность результатов проектирования для предприятия (организации); отражение новых разработок и исследований в области построения АСОИУ.

В соответствии с квалификационной характеристикой выпускников по направлениям 230100 и 010500 выполнение КП осуществляется в рамках следующих направлений:

1. Автоматизированные системы (подсистемы) обработки информации и управления (АСОИУ).
2. Информационно-моделирующие системы (подсистемы).
3. Автоматизированные системы экспериментальных исследований.
4. Автоматизированные системы (подсистемы) научных исследований (АСНИ).
5. Геоинформационные системы.
6. Системы распознавания зрительных образов и анализа изображений.
7. Автоматизированные обучающие системы.

Особую роль при подготовке специалистов в области информационных технологий (ИТ) играют навыки проектирования и создания компьютеризированного интегрированного производства. В этой связи при выборе темы КП предпочтение следует отдавать проектированию автоматизированных информационных систем (или их составляющих), используемых в жизненном цикле промышленных изделий [78]:

- CAE (Computer Aided Engineering) – системы автоматизированных расчетов и анализа;
- CAD (Computer Aided Design) – системы автоматизированного проектирования;
- CAM (Computer Aided Manufacturing) – автоматизированные системы технологической подготовки производства;
- PDM (Product Data Management) – системы управление проектными данными;
- ERP (Enterprise Resource Planning) – системы планирования и управления предприятием;
- MRP (Manufacturing Requirement Planning) – системы планирования производством;
- MES (Manufacturing Execution System) – производственные исполнительные системы;
- SCM (Supply Chain Management) – системы управления цепочками поставок;
- CRM (Customer Relationship Management) – системы управления взаимоотношениями с заказчиками;
- SCADA (Supervisory Control And Data Acquisition) – системы диспетчерского управления производственными процессами;
- CNC (Computer Numerical Control) – компьютерное числовое управление;
- S&SM (Sales and Service Management) – системы управления продажами и обслуживанием;
- CPC (Collaborative Product Commerce) – системы совместного электронного бизнеса.

Студентам, по согласованию, могут быть предложены индивидуальные, нестандартные задания по теме данной дисциплины, связанные с выполнением госбюджетных и хоздоговорных НИР и ОКР, проводимых на кафедре, включая разработку лабораторных стендов и товаров народного потребления. В свою очередь, студент может самостоятельно предложить тему КП, отличающуюся повышенной актуальностью, связанную с выполнением НИРС или предстоящей выпускной квалификационной работы. Тема КП утверждается руководителем в установленные кафедрой сроки.

3. СОДЕРЖАНИЕ И ОБЪЕМ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ ПО КУРСОВОМУ ПРОЕКТУ

На страницах пояснительной записки (ПЗ) проводят анализ предметной области и объекта автоматизации, анализ требований к проектируемой системе, разрабатывают концепцию системы и эскизный проект. Выбор каждого проектно-технологическое решение должен проводиться из нескольких альтернатив и должен быть тщательно обоснован. При этом рекомендуется на страницах ПЗ проводить анализ альтернативных вариантов архитектуры системы, программно-технологических платформ и подходов, а результаты проведенного анализа помещать в тот или иной раздел соответствующего приложения.

Содержание ПЗ по КП должно отражать результаты создания АСОИУ до стадии «4. Эскизный проект» включительно по ГОСТ 34.601-90. ПЗ должна включать в свой состав (в скобках указан рекомендуемый объем в страницах):

- титульный лист (1 стр.);
- аннотацию (0,5 стр.);
- задание на курсовой проект (1 стр.);
- содержание (1 стр.);
- список принятых сокращений (при необходимости) (1 стр.);
- введение (2-3 стр.);
- проведение обследования объекта автоматизации (5-10 стр.);
- формирование требований к проектируемой системе (10-15 стр.);
- разработку концепции АСОИУ (10-20 стр.);
- техническое задание (3-7 стр.);
- эскизный проект (10-15 стр.);
- заключение (1 стр.);
- список литературы (1 стр.);
- приложения.

Рекомендуемый объем пояснительной записки составляет 45-60 страниц печатного текста на листах писчей бумаги формата А4 (210х297 мм), на одной стороне листа. При необходимости, в приложениях допускается применение листов формата А3 (297х420 мм).

4. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ОСНОВНЫХ РАЗДЕЛОВ КУРСОВОГО ПРОЕКТА

4.1. ТИТУЛЬНЫЙ ЛИСТ

Титульный лист является первым листом пояснительной записки. Образец его оформления приведен в приложении А.

4.2. ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ

Задание на курсовой проект является исходным документом для проектирования. Оно составляется на специальном бланке (см. приложение Б) и содержит следующие данные:

- тему курсового проекта;
- исходные данные: назначение проектируемой АСОИУ и ее основные характеристики;
- указания по содержанию пояснительной записки;
- сроки выдачи задания и представления выполненной работы.

4.3. АННОТАЦИЯ

Аннотация содержит сведения об объеме ПЗ, количестве иллюстраций, таблиц, использованных литературных источников и отражает: объект проектирования; предмет проектирования; используемые в ходе проектирования методы; краткое содержание разделов пояснительной записки с указанием отличительных особенностей работы; полученные результаты и их оценку. Необходимо отметить, что под объектом курсового проекта, как правило, понимается автоматизация технологического процесса, рассматриваемого в работе. Предмет курсового проекта представляет собой приложение информационных технологий к решению конкретной прикладной задачи, относящейся к объекту проектирования, и обуславливающей актуальность заданной темы.

Аннотация составляется на заключительном этапе оформления ПЗ и снабжается основной надписью для текстовых документов по ГОСТ 2.104-2006 (Форма 2) [64]. Пример оформления аннотации приведен в приложении В.

4.4. СОДЕРЖАНИЕ

Содержание приводится в пояснительной записке после аннотации и включает наименования всех разделов и подразделов с указанием номера листа, с которого начинается каждый из них. Первый лист содержания необходимо снабдить основной надписью для текстовых документов (ГОСТ 2.104-2006 последующие листы – Форма 2а [64]). Пример оформления первого листа содержания (номера страниц намеренно не проставлены) приведен в приложении Г. Структура содержания, приведенная в приложении Г, при необходимости может быть детализирована путем введения дополнительных подразделов в ту или иную главу.

4.5. ВВЕДЕНИЕ

Во введении необходимо сформулировать проблему, цель и задачи курсового проекта, актуальность, провести краткий обзор известных программно-технологических решений по теме курсового проекта.

В обзоре основное внимание следует уделить типовым структурным, архитектурным и программным решениям, используемым в системах, аналогичных проектируемой; уровню развития технологий проектирования, включающих в свой состав методологию проектирования и системы управления проектом.

Основными литературными источниками, рекомендуемыми для обзора, являются: [10, 12, 14, 15, 26, 27, 28, 29, 30, 34, 38, 41, 44, 46, 48, 55, 56, 62, 79, 82, 83, 91, 94, 98, 103] (типовые структурные, архитектурные и программные решения); [18, 23, 93] (системы управления проектом); [2, 4, 9, 11, 17, 47, 61, 88] (проектирование систем доступа к данным), а также сборники международных конференций: «Advanced Software Engineering: Expanding the Frontiers of Software Technology» [1], «Empirical Software Engineering Issues Critical Assessment and Future Directions» [5], «Fundamental Approaches to Software Engineering» [6], «Information Systems Development» [7], «Programming Languages and Systems» [8], «Web Information Systems Engineering» [9] за последние 5-7 лет, в которых, как правило, публикуется описание новейших разработок и технологий в области создания АСОИУ. Вопросам проектирования современных информационно-управляющих и моделирующих систем промышленного назначения посвящена обширная литература, наиболее полно перечисленная в [16, 19, 49].

При обосновании актуальности разработки следует сравнить основные характеристики известных аналогов с требованиями задания на курсовой проект и на основании этого сравнения сделать заключение о целесообразности разработки системы.

Следует заметить, что, как правило, введение составляется в конце работы над курсовым проектом, когда известны основные результаты, что позволяет конкретизировать цель и задачи.

4.6. ПРОВЕДЕНИЕ ОБСЛЕДОВАНИЯ ОБЪЕКТА АВТОМАТИЗАЦИИ

4.6.1. Сбор и анализ данных об объекте автоматизации

Сбор и анализ данных об объекте автоматизации, включая документооборот, осуществляется посредством составления и описания различных структурных схем, отличающихся типами элементов и связей между ними. ГОСТ 24.103-84 [66] предусматривает использование следующих видов структур: (i) функциональная (элементы - функции, задачи, операции; связи - информационные); (ii) тех-

ническая (элементы-устройства; связи - линии связи); (iii) организационная (элементы - коллективы людей и отдельные исполнители; связи - информационные, соподчинения и взаимодействия); (iv) алгоритмическая (элементы - алгоритмы; связи - информационные); (v) программная (элементы - программные модули; связи - информационные и управляющие); (vi) информационная (элементы - формы существования и представления информации в системе; связи - операции преобразования информации в системе).

В процессе сбора и анализа данных об объекте автоматизации необходимо использовать технологии системного анализа, позволяющие с обобщенных позиций изучить проблемную ситуацию, требующую для своего решения привлечения информационных технологий. Согласно концепции системного анализа решение проблемы в большинстве случаев связывается с реализацией «**улучшающего вмешательства**», т.е. воздействия, которое положительно оценивается хотя бы одним субъектом и неотрицательно всеми остальными [96]. Под субъектами проблемной ситуации понимаются все ее участники (в терминологии системного анализа - «стейкхолдеры»): организации, предприятия, учреждения. Изменения будут вноситься в интересах одних из них («проблемосодержащих»), осуществляться за счет ресурсов других («проблеморазрешающих») и как-то сказываться на остальных. В этой связи, при проведении обследования объекта автоматизации составляется по возможности полный список стейкхолдеров, опрос которых впоследствии позволит учесть все многообразие требований к системе.

Кроме того, на первом шаге обследования необходимо построить и описать модель «черного ящика» системы, представляющую собой перечень входов и выходов системы, и, предназначенную, с одной стороны, для указания границ проектируемой системы, с другой стороны, для фиксирования значимых с точки зрения заказчика входных и выходных каналов передачи данных.

Необходимо указать какими данными и в каком формате пользователь должен располагать для полноценного взаимодействия с системой, а также, какие данные, и в каком формате проектируемая система должна возвращать в виде результата в соответствии с заданием.

Например, если проектируемая система предназначена, кроме всего прочего, для формирования отчетов, то необходимо, поместив типовую форму отчета в приложение к ПЗ, описать его поля, указав их тип и назначение.

Затем модель «черного ящика» подвергают декомпозиции, получая иерархическое дерево подсистем или модель состава системы.

В ПЗ необходимо провести описание декомпозиции проектируемой системы согласно методологии структурного анализа и проектирования SADT (Structured Analysis & Design Technique) до уровня подсистем, выполняющих отдельно взятые функции.

При этом рекомендуется структурно-функциональные IDEF0-диаграммы верхнего уровня декомпозировать с применением диаграммам IDEF3, которые позволяют представить не только информационные потоки и взаимоотношения между процессами обработки информации, но и объекты, являющиеся частью этих процессов и, следовательно, обладают более богатой по сравнению с IDEF0 нотацией [84]. Построение структурно-функциональных диаграмм необходимо выполнять с использованием той или иной CASE-системы (например, AllFusion Process Modeler (BPWin), входящей в состав пакета AllFusion Modeling Suite (Computer Associates) [84]).

Вопросам анализа предметной области и предварительного моделирования ИС посвящен широкий спектр литературных источников, однако в рамках курсового проекта достаточно воспользоваться методиками, изложенными в [31, 37, 87]. Базовые концепции и приемы системного анализа изложены в двух классических работах [90, 96].

Комплект IDEF0(IDEF3)-диаграмм, оформленных согласно Р 50.1.028-2001 [92], приводят в приложении к ПЗ. Пример оформления контекстной IDEF0-диаграммы и одной диаграммы декомпозиции приведен в приложении Д методических указаний.

4.6.2. Разработка обоснования на создание АСОИУ (ИМС)

4.6.2.1 Формулирование цели и задач создания АСОИУ (ИМС)

В подразделе «Формулирование цели и задач создания АСОИУ (ИМС)» приводят развернутую формулировку цели создания системы, а также наименования и требуемые значения технических, технологических, производственно-экономических или других показателей объекта автомати-

зации, которые должны быть достигнуты в результате создания системы с **указанием критериев оценки достижения цели** [71].

Здесь же **приводят подробный перечень задач**, которые необходимо решить для достижения поставленной цели, а также **описывают выбор и обоснование состава процессов, подлежащих автоматизации**.

Для этого все функции проектируемой системы, перечисленные в разделе «Сбор и анализ данных об объекте автоматизации», разделяют на две группы: (i) функции, которые будут автоматизированы в ходе выполнения проекта; (ii) функции, которые останутся неавтоматизированными с обоснованием причин.

Каждая функция системы, подлежащая автоматизации, в соответствии с заданием на КП, должна снабжаться исчерпывающим описанием, включающим в свой состав расширенное определение ее назначения, а также анализ входных и выходных данных.

Для формулирования цели создания рассматриваемой системы полезно провести систематизацию данных, полученных в результате опроса стейкхолдеров. При этом необходимо учесть следующее:

- существует опасность неполного перечисления целей;
- цели, объявленные стейкхолдером, могут отличаться от его истинных целей;
- цели могут быть подменены средствами. Цели имеют древовидную структуру и каждый элемент является целью для более низкого уровня и средством для более высокого. Цель самого высокого уровня формулируют как миссию, которая отражает интересы всех стейкхолдеров. Далее определяют идеалы – результаты, которые считаются недостижимыми, но приближение к которым возможно, затем цели – результаты, которые не предполагается достичь в рамках курсового проекта, но к которым рассчитывают приблизиться и задачи – результаты, которые предполагается получить по завершению работы над проектом.

При формулировании цели необходимо придерживаться следующих рекомендаций:

1. цель должна быть достижимой при наличии финансовых, материальных и временных ресурсов;
2. цель должна быть конкретной, т.е. любое продвижение к цели должно вносить вклад в решение именно данной проблемы, а не какой-нибудь другой;
3. цель должна быть измеримой, т.е. формулировка цели должна предполагать возможность отслеживать процесс движения к цели путем измерений.

Рекомендации по формулированию целей в достаточном для выполнения курсового проекта объеме изложены в работе [96].

4.6.2.2 Предварительный выбор и обоснование состава автоматизируемых функций системы

В данном разделе функции проектируемой системы, которые планируется автоматизировать в рамках проекта, анализируются с точки зрения пользователя. Цель такого анализа состоит в том, чтобы, с одной стороны, прояснить взаимосвязи между функциями проектируемой системы, с другой стороны подготовить исходные данные для предварительного расчета затрат на создание системы.

Международная группа пользователей функционального измерения (IFPUG – International Function Point Users Group) [25] опубликовала критерии, по которым все функции системы разделяют на пять групп в зависимости от принадлежности к одной из следующих информационных характеристик:

- Внешний ввод – элементарный процесс, перемещающий данные из внешней среды в приложение. Данные могут поступать по каналам связи, от пользователя с экрана ввода или из другого приложения. Данные могут использоваться для обновления внутренних логических файлов и могут содержать как управляющую, так и деловую информацию. Управляющие данные не должны модифицировать внутренний логический файл.
- Внешний вывод – элементарный процесс, перемещающий данные, вычисленные в приложении, во внешнюю среду. Кроме того, в этом процессе могут обновляться внутренние логические файлы. Данные создают отчеты или выходные файлы, посылаемые другим приложениям. Отчеты и файлы создаются на основе внутренних логических файлов и внешних интерфейсных файлов. Дополнительно этот процесс может использовать вводимые данные, их образуют критерии поиска и параметры, не поддерживаемые внутренними логическими файлами. Вводимые данные поступают извне, но носят временный характер и не сохраняются во внутреннем логическом файле.
- Внешний запрос – элементарный процесс, работающий как с вводимыми, так и с выводимыми данными. Его результат – данные, возвращаемые из внутренних логических файлов и внешних

интерфейсных файлов. Входная часть процесса не модифицирует внутренние логические файлы, а выходная часть не несет данных, вычисляемых приложением (в этом состоит отличие запроса от вывода).

- Внутренний логический файл – распознаваемая пользователем группа логически связанных данных, которая размещена внутри приложения и обслуживается через внешние вводы.
- Внешний интерфейсный файл – распознаваемая пользователем группа логически связанных данных, которая размещена внутри другого приложения и поддерживается им. Внешний файл данного приложения является внутренним логическим файлом в другом приложении.

Вводы, выводы и запросы относят к категории транзакций. В данном контексте согласно определению IFPUG под транзакцией понимается элементарный процесс, различаемый пользователем и перемещающий данные между внешней средой и программным приложением. В своей работе транзакции используют внутренние и внешние файлы [55, 89].

Каждой информационной характеристике ставится в соответствие сложность. Для этого характеристике назначается низкий, средний или высокий ранг с соответствующей числовой оценкой. Для транзакций ранжирование основано на количестве ссылок на файлы и количестве типов элементов данных. Для файлов ранжирование основано на количестве типов элементов-записей и элементов данных, входящих в файл. Тип элемента-записи — подгруппа элементов данных, распознаваемая пользователем в пределах файла. Тип элемента данных — уникальное не рекурсивное (неповторяемое) поле, распознаваемое пользователем.

Под элементами данных, как правило, понимают: для внешних вводов – поля ввода данных, сообщения об ошибках, вычисляемые значения, кнопки; для внешний выводов – поля данных в отчетах, вычисляемые значения, сообщения об ошибках, заголовки столбцов, которые читаются из внутреннего файла; для внешних запросов – вводимые элементы: поле, используемое для поиска, щелчок мыши, – выводимые элементы: отображаемые на экране поля.

Дополнительные примеры элементов данных, правила учета элементов графического пользовательского интерфейса, а также порядок учета сообщений приведены в [55:139-140, 89:24-25].

В данном разделе на страницах ПЗ каждую функцию системы, выделенную в главе «Сбор и анализ данных об объекте автоматизации», необходимо сопоставить с той или иной информационной характеристикой.

С этой целью диаграмму декомпозиции, полученную при проведении сбора и анализа данных об объекте автоматизации, рекомендуется представить в виде диаграммы потоков данных (Data Flow Diagrams – DFD), поскольку именно DFD позволяет отражать функции обработки информации (работы); объекты (документы, сотрудники и проч.), участвующие в обработке информации; внешние ссылки (external reference), обеспечивающие интерфейс с внешними объектами, и таблицы для хранения данных (data store).

Результаты проведенного анализа будут использоваться далее в разделе «Оценка затрат и предварительный расчет ожидаемой эффективности АСОИУ».

4.6.2.3 Оценка затрат и предварительный расчет ожидаемой эффективности АСОИУ (ИМС)

Заказчика на протяжении всего процесса работы над проектом интересует его стоимость. Процесс оценки стоимости, как правило, начинается одновременно со стартом проекта и продолжается даже на стадии написания программного кода. На ранних стадиях проекта необходимо делать оценку несколькими разными способами независимо, а затем сравнивать результаты.

В рамках курсового проекта предлагается воспользоваться хорошо зарекомендовавшей себя методологией оценивания функционального размера (FP – Functional Points), которая заключается в единообразном измерении всех возможностей приложения и выражении размера приложения в виде числа, которое затем можно использовать для определения числа строк кода, стоимости и сроков проекта.

Для вычисления функционального размера определяют ранг для каждой информационной характеристики, описанной в разделе «Предварительный выбор и обоснование состава автоматизируемых функций системы». В частности, в табл. 1 приведены данные для определения ранга и оценки сложности внешних вводов (числовая оценка указана в круглых скобках). Из данных, приведенных в табл. 1, следует, что, например, внешнему вводу, который ссылается на 2 файла и имеет 7 элементов данных, назначается средний ранг и оценка сложности – 4).

Таблицы для определения рангов прочих информационных характеристик приведены в [55, 89].

После определения ранга всех информационных характеристик системы приступают к расчету функционального размера. Исходные данные для расчета общего количества информационных характеристик сводят в табл. 2. Полученные в каждой строке значения суммируются, давая итоговое значение для данной характеристики. Итоговые значения затем суммируются по вертикали, формируя общее количество информационных характеристик.

Таблица 1

Ранг и оценка сложности внешних вводов

Ссылки на файлы	Элементы данных		
	1-4	5-15	>15
0-1	Низкий (3)	Низкий (3)	Средний (4)
2	Низкий (3)	Средний (4)	Высокий (6)
>2	Средний (4)	Высокий (6)	Высокий (6)

Таблица 2

Исходные данные для вычисления общего количества информационных характеристик

Имя характеристики	Количество			
	Низкий	Средний	Высокий	Итого
Внешние вводы	_ x 3 =	_ x 4 =	_ x 6 =	
Внешние выводы	_ x 4 =	_ x 5 =	_ x 7 =	
Внешние запросы	_ x 3 =	_ x 4 =	_ x 6 =	
Внутренние логические файлы	_ x 7 =	_ x 10 =	_ x 15 =	
Внешние интерфейсные файлы	_ x 5 =	_ x 7 =	_ x 10 =	
Общее количество				

Функциональный размер вычисляется по следующей формуле [55, 89]:

$$FP = \text{Общее количество} \times (0{,}65 + 0{,}01 \times \sum_{i=1}^{14} F_i), \qquad (1)$$

где F_i – коэффициенты сложности. Каждый коэффициент F_i в (1) может принимать следующие значения: 0 – нет влияния, 1 – случайное, 2 – небольшое, 3 – среднее, 4 – важное, 5 – основное. Значение каждого i-ого коэффициента определяется эмпирически в результате ответа на 14 вопросов, которые характеризуют системные параметры приложения (см. табл. 3). В [57:439-444] приводится детальное описание системных параметров, перечисленных в таблице 3, и даются рекомендации по определению величины сложности для каждого из них.

Таблица 3

Определение системных параметров приложения

№	Системный параметр	Описание
1	Передача данных	Сколько средств связи требуется для передачи или обмена информацией с приложением или системой?
2	Распределенная обработка данных	Как обрабатываются распределенные данные и функции обработки?
3	Производительность	Нуждается ли пользователь в фиксации времени ответа или производительности?
4	Распространенность используемой конфигурации	Насколько распространена текущая аппаратная платформа, на которой будет выполняться приложение?
5	Скорость транзакций	Как часто выполняются транзакции? (каждый день, каждую неделю, каждый месяц)
6	Оперативный ввод данных	Какой процент информации надо вводить в режиме онлайн?
7	Эффективность работы конечного пользователя	Приложение проектировалось для обеспечения эффективной работы конечного пользователя?
8	Оперативное обновление	Как много внутренних файлов обновляется в онлайновой транзакции?
9	Сложность обработки	Выполняет ли приложение интенсивную логическую или математическую обработку?

№	Системный параметр	Описание
10	Повторная используемость	Приложение разрабатывалось для удовлетворения требований одного или многих пользователей?
11	Легкость инсталляции	Насколько трудны преобразование и инсталляция приложения?
12	Легкость эксплуатации	Насколько эффективны и/или автоматизированы процедуры запуска, резервирования и восстановления?
13	Разнообразные условия размещения	Была ли спроектирована, разработана и поддержана возможность инсталляции приложения в разных местах для различных организаций?
14	Простота изменений	Была ли спроектирована, разработана и поддержана в приложении простота изменений?

Для оценивания затрат труда и продолжительности проекта рекомендуется использовать конструктивную модель стоимости COCOMO (Constructive Cost Model), предложенную в 1981г. Барри Боэмом [55, 89]. Подмодели COCOMO могут применяться к трем типам программных проектов. По терминологии Боэма, их образуют:

- распространенный тип – небольшие программные проекты, над которыми работает небольшая группа разработчиков с хорошим стажем работы, устанавливаются мягкие требования к проекту;
- полунезависимый тип – средний по размеру проект, выполняется группой разработчиков с разным опытом, устанавливаются как мягкие, так и жесткие требования к проекту;
- встроенный тип – программный проект разрабатывается в условиях жестких аппаратных, программных и вычислительных ограничений.

Уравнения базовой подмодели COCOMO имеют вид:

$$E=a\ x(KLOC)^b\ [чел\text{-}мес]; \quad (2)$$

$$D = c\ x\ (E)^d\ [мес], \quad (3)$$

где E – затраты в человеко-месяцах, D – время разработки, KLOC – количество тысяч строк в программном продукте. Коэффициенты a, b, c, d определяются по табл. 4.

Таблица 4

Коэффициенты для базовой подмодели COCOMO

Тип проекта	a	b	c	d
Распространенный	2,4	1,05	2,5	0,38
Полунезависимый	3,0	1,12	2,5	0,35
Встроенный	3,6	1,20	2,5	0,32

Для определения числа строк кода (KLOC) используют стандартные таблицы по функциональному размеру. Например, в таблице 2.13 [89:28] одной единице функционального размера соответствует 53 строки программного кода на языке программирования Java, 64 строки на C++, 29 строки на Delphi Pascal.

Из (2) следует, что согласно модели Боэма продолжительность проекта не зависит от количества привлекаемых для его реализации специалистов, а зависит от функционала и типа системы.

Модель Боэма проста в использовании и широко распространена, однако позволяет получить весьма грубые оценки. При необходимости использования более точных оценок затрат и времени разработки рекомендуется обратиться к литературным источникам [18, 21, 23, 32], а также материалам сайтов [20, 25].

Полученные оценки позволят скорректировать сроки выполнения проекта и затраты на его реализацию. Обсуждение этих результатов совместно с заказчиком может повлиять на итоговую версию договора.

4.6.2.4 Сбор и анализ данных о зарубежных и отечественных аналогах

Проводят более детальный, чем во введении поиск, анализ и описание архитектуры, технологических и программных решений систем, аналогичных проектируемой. Цель этого подраздела: подтвердить актуальность проекта с учетом предъявляемых требований и наложенных ограничений, а также выявить перечень стандартных компонентов, модулей, пакетов и т.д., использование которых в проекте без существенных модификаций позволило бы сократить ресурсы на их повторное проектирование.

Необходимо показать какие наработки, приемы современных информационных технологий будут использованы в проекте, а также какие программные компоненты (классы, интерфейсы и т.д.), многократно проверенные (протестированные) на практике, могут составить основу проекта.

Подходы проектирования информационных систем (ИС), основанные на повторном использовании компонентов, к настоящему времени сформировались в несколько крупных направлений и широко обсуждаются в литературе [42, 62, 101].

На основании проведенного анализа посредством сравнения затрат на проектирование заданной системы и стоимости систем аналогичного класса делается заключение о целесообразности проектирования АСОИУ с параметрами, указанными в задании на КП.

4.7. ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОЙ АСОИУ (ИМС)

4.7.1 Обследование АСОИУ (ИМС)

4.7.1.1. Исходные материалы и документы по созданию АСОИУ (ИМС)

В данном параграфе проводят систематизацию нормативных документов, определяющих основные понятия в области автоматизированных систем (АС) и, устанавливающих термины и определения [69], основные положения [66], общие требования [67], процессы жизненного цикла [73], стадии создания АС [68, 70] и т.д.

В ПЗ приводят диаграмму, на которой изображают структуру нормативных документов, использованных при подготовке проекта, а также нормативно-правовых документов, относящихся к организации, эксплуатации и обслуживанию технологического процесса, автоматизации которого посвящен проект.

При анализе нормативных документов необходимо иметь в виду следующее заключение, сделанное специалистами в области стандартизации разработки программных средств [54:61]: «Процессы создания автоматизированных систем (АС), в состав которых входит ПО, регламентированы стандартами ГОСТ 34.601-90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания», ГОСТ 34.602-89 «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы» и ГОСТ 34.603-92 «Информационная технология. Виды испытаний автоматизированных систем». Однако процессы создания ПО для современных распределенных АИС, функционирующих в неоднородной среде, в этих стандартах отражены недостаточно, а отдельные их положения явно устарели. В результате для каждого серьезного проекта АИС приходится создавать комплекты нормативных и методических документов, регламентирующих процессы создания конкретного прикладного ПО, поэтому в отечественных разработках целесообразно использовать современные международные стандарты» (выделено автором). В этой связи, при выполнении курсового проекта рекомендуется, основываясь на структуре ГОСТов 34.xxx, содержательную часть разделов рассматривать, ориентируясь на стандарты, опубликованные Институтом инженеров по электротехнике и радиоэлектронике (Institute of Electrical and Electronics Engineers (IEEE)) [43].

Ввиду ограниченного объема времени, выделенного на курсовое проектирование, номенклатура разрабатываемой технической документации отличается от предусмотренной действующими нормативными документами и оговаривается в задании на курсовой проект. Минимальный комплект документации, который необходимо разработать в ходе выполнения курсового проекта включает в свой состав следующие основные документы [43, 55]:

1. план управления конфигурациями программного обеспечения (Software Configuration Management Plan – SCMP) в соответствии со стандартом IEEE 828-1990;
2. план контроля качества программного обеспечения (Software Quality Assurance Plan – SQAP) в соответствии со стандартом IEEE 730-1989;
3. план управления программным проектом (Software Project Management Plan – SPMP) в соответствии со стандартом IEEE 1058.1-1987;
4. спецификация требований к программному обеспечению (Software Requirements Specification – SRS) в соответствии со стандартом IEEE 830-1993;
5. техническое задание (ТЗ) в соответствии со стандартом ГОСТ 34.602-89;
6. проектная документация программного обеспечения (Software Design Document – SDD) в соответствии со стандартом IEEE 1016-1987.

4.7.1.2. Обоснование выбора модели жизненного цикла АСОИУ (ИМС)

Ключевую роль в успешной разработке многокомпонентных программно-аппаратных систем играет **организация** стадий создания системы во времени. Модели организации жизненного цикла ИС, появившиеся в конце 80-х, начале 90-х годов (каскадная (водопадной) и спиральная (итерационная)), в настоящее время находят весьма узкое применение. Вместе с тем, существует несколько современных концепций организации жизненного цикла, которые необходимо проанализировать в разделе «Обоснование выбора модели жизненного цикла АСОИУ».

Особое внимание следует уделить Унифицированному процессу разработки программного обеспечения (Unified Software Development Process - USDP), разработанному сотрудниками компании IBM Corporation [55, 56, 103].

Унифицированный процесс представляет собой попытку объединения водопадной и итеративной моделей ЖЦ. При этом, ЖЦ при использовании USDP разделен на 4 фазы:

1. анализ и планирование требований: (i) создается упрощенная модель вариантов использования, содержащая наиболее критичные с точки зрения реализации прецеденты; (ii) создается пробный вариант архитектуры, содержащей наиболее важные подсистемы; (iii) проводится идентификация и определение приоритетов рисков; (iv) планируется фаза проектирования; (v) грубо оценивается весь проект в целом;
2. проектирование: детально описываются большинство вариантов использования и разрабатывается архитектура системы. В конце фазы проектирования менеджер проекта выполняет подсчет ресурсов, необходимых для завершения проекта. Необходимо ответить на вопрос: достаточно ли проработаны варианты использования, архитектура и план, чтобы можно было бы давать контрактные обязательства на выполнение работы и переходить к подготовке и подписанию «Технического задания»?;
3. построение – создание продукта. В конце этой фазы продукт включает в себя все варианты использования, которые разработчики и заказчик решили включить в текущий выпуск;
4. внедрение – выпуск продукта. Проводится тестирование бета-версии продукта отделом тестирования компании и одновременно организуется пробная эксплуатация системы пользователями. После этого разработчики исправляют обнаруженные ошибки и вносят некоторые из предложенных улучшений в главный выпуск, подготавливаемый для широкого распространения.

Каждая фаза USDP может включать в свой состав одну или несколько итераций в зависимости от размера проекта (см. рис. 1). На протяжении каждой итерации может выполняться 5 основных и некоторое количество дополнительных рабочих потоков. К основным рабочим потокам в USDP относятся: определение требований (ОТ); анализ (А); проектирование (П); реализация (Р); тестирование (Т). К дополнительным рабочим потокам могут относиться: работы по обеспечению качества (К), документирование (Д), управление проектом (У), управление конфигурацией (УК), создание и управление инфраструктурой (И) и другие.

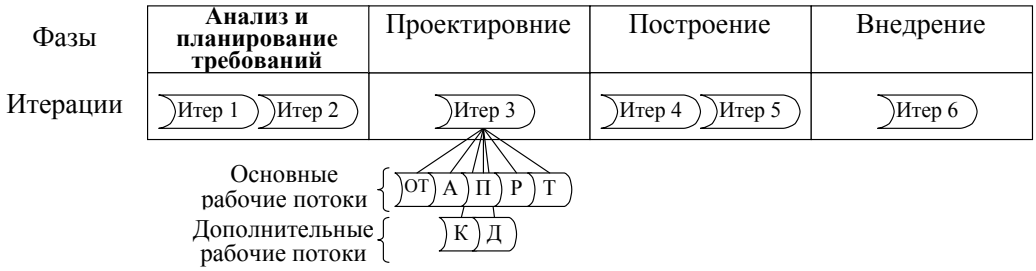

Рис. 1. Модель ЖЦ согласно унифицированного процесса разработки ПО

Не смотря на популярность унифицированного процесса разработки программного обеспечения, ставшего в настоящее время классическим, появляются новые подходы к организации и управлению процессом разработки, объединяющие в себе накопленный при использовании USDP опыт, а также использующие ряд новых инструментов и идей (например, ICONIX Process [40], экстремальное программирование (Extreme Programming) [50, 52], быстрая разработка ПО (Agile Software Development) [3], интегрированные методики [26, 44], технологии мультиагентных систем [22] и др.).

На страницах ПЗ необходимо с использованием литературных источников [3, 22, 26, 40, 44, 50, 52] проанализировать основные концепции процесса создания ПО, выбранного в качестве базы для выполняемого курсового проекта.

Принятая к использованию в проекте модель организации жизненного цикла системы впоследствии должна найти отражение при составлении плана управления программным проектом (см. п. 4.7.1.5).

4.7.1.3. План управления конфигурациями

Управление документацией программного проекта требует значительных организационных навыков, поскольку документация крупного проекта представляет собой сложноорганизованную систему документов с множеством перекрестных связей, которая подвержена непрерывным изменениям в ходе выполнения проекта.

Управление документацией подразумевает поддержание ее полноты, согласованности и включает в себя управление конфигурациями.

Полнота документации обуславливается перечнем стандартов, использованных при разработке проекта.

Согласованность документации означает, что набор документов не содержит внутренних противоречий.

Обеспечение непротиворечивости достигается за счет создания целостной документации, т.е. такой документации, в которой каждая спецификация располагается только в одном месте.

Решение проблемы несогласованности возможно за счет использования гипертекстовых документов с поддержкой перекрестных ссылок.

Поддержка конфигурации – это координация различных версий и частей документации и программного кода. Для отслеживания частей проекта необходимо определить их границы, т.е. ввести элементы конфигурации под которыми, как правило, понимают классы, значимые наборы данных, редко - отдельные функции. Существуют специальные программные средства для управления конфигурацией (например, Microsoft Visual SourceSafe, Microsoft Visual Studio Team Foundation Server, IBM Rational ClearCase, Subversion и др.). Обычно системы управления конфигурациями удовлетворяют следующим минимальным требованиям [55, 93]:

- возможность определения элементов конфигурации;
- отказ в праве на модификацию для предотвращения одновременной работы более чем одного человека над элементом конфигурации;
- авторизация доступа с правом модификации;
- процедура включения в проект;
- запись о предыдущих группировках согласующихся элементов конфигурации.

На страницах ПЗ необходимо с использованием литературных источников [53, 60, 93] проанализировать возможности, достоинства и недостатки наиболее распространенных систем управления конфигурацией и выбрать одну из них по следующей общей методике.

Приведенная далее последовательность шагов может быть использована при решении задачи выбора любой инструментальной системы (языка программирования, СУБД и т.д.) из нескольких альтернатив:

1. Выявляются базовые показатели выбираемой системы, значимые при проектировании заданной АСОИУ с учетом ее особенностей, ограничений, ресурсов и т.д.
2. Все показатели сводятся в таблицу (см. табл. 5), в которой на основании экспертных оценок каждому показателю назначается весовой коэффициент V_j (например, от 1 до 10), характеризующий значимость данного показателя по сравнению с остальными. Сумма значений всех весовых коэффициентов должна равняться его верхней границе (в рассматриваемом примере - 10).
3. С использованием данных, полученных из литературных источников и/или от экспертов, по каждому *j*-ому показателю для каждой *i*-ой системы определяется степень полезности $U_{i,j}$ (от 1 – минимальная, до 10 - максимальная). Например, степень полезности системы управления конфигурациями, стоимость которой является сравнительно высокой, может равняться 1, тогда как степень полезности свободно-распространяемой системы будет равна 10.
4. Для каждой *i*-ой сравниваемой системы вычисляется значение оценочной функции по формуле: $F_i = V_1 \times U_{i,1} + V_2 \times U_{i,2} + \ldots = \sum V_j \times U_{i,j}$
5. На основании значения оценочной функции делается вывод о целесообразности использования той или иной системы в данном проекте при учете выбранных критериев и заданных ограничений. Та система, для которой значение оценочной функции окажется больше, является лучшей из числа сравниваемых альтернатив.

В приложении к ПЗ необходимо привести план управления конфигурациями разрабатываемой системы в соответствии со стандартом IEEE 828-1990. Пример составления плана см. в [55]. Оглавление плана управления конфигурациями приведено в приложении Е и включает в качестве примера типовое описание некоторых разделов, которое необходимо адаптировать в соответствии с содержанием выполняемого проекта. Примечание к тому или иному разделу приведено в квадратных скобках курсивным шрифтом.

Таблица 5

Процедура выбора системы из нескольких альтернатив

j	Показатель	Вес V_j (1-10)	Преимущество системы №1, $U_{1,j}$ (от 1 - минимальное до 10 - максимальное)	Преимущество системы №2, $U_{2,j}$ (от 1 - минимальное до 10 - максимальное)	Преимущество системы №n, $U_{n,j}$
1	Трудоемкость установки и настройки	1	2	6	5
2	Удобство сетевого взаимодействия	4	6	4	8
3	Стоимость	4	10 (бесплатная)	1 (очень дорогая)	5
	…	…	…	…	…
	ИТОГ	10	F_1=1x2+4x6+4x10+…	F_2=1x6+4x4+4x1+…	F_n=1x5+4x8+4x5+…

4.7.1.4. План контроля качества программного обеспечения

Переход от штучной разработки АСОИУ к промышленному производству обусловил повышение требований к качеству создаваемых систем. В настоящее время существует несколько стандартов, связанных с оценкой качества этих процессов, например:

- Международные стандарты серии ISO 9000;
- Capability Maturity Model (CMM) – модель зрелости (совершенствования) процессов создания ПО, предложенная SEI (Software Engineering Institute – Институт технологий программного обеспечения);
- IEEE 739-1989 Software Quality Assurance Plan (SQAP) – план контроля качества программного обеспечения.

Управление качеством при проектировании программного продукта (определение необходимого уровня качества и выработка стратегии достижения заданного уровня) предполагает применение совокупности различных подходов (инспектирование, формальные методы, тестирование, методы управления проектом) в основе которых лежит использование численных оценок некоторых характеристик проекта. Указанные характеристики получили название метрик.

Метрики, которые необходимо использовать при выполнении проекта:

- объем выполненной работы, измеренный в физических единицах (стр. документации);
- время, затраченное на выполнение единицы работы (например, на 1 стр. или 10 стр. документации);
- степень дефектности (число дефектов на страницу документации).

Предварительные или желаемые значения метрик заранее прогнозируются, а затем сравниваются с полученными результатами.

Когда результаты обнаружения дефектов сравниваются с нормами организации, происходит оценка всего процесса создания системы в целом, а не только конкретного проекта. Накопленные данные о дефектах на каждой стадии сводятся в таблицу так, как показано, например, в табл. 6.

Таблица 6

Среднее количество дефектов на каждой стадии проекта

Стадии, на которых были обнаружены дефекты	Стадии, содержащие дефекты (в данном проекте / норма)		
	Формирование требований	Техническое задание	Эскизный проект
Формирование требований	2/5		
Техническое задание	0,5/1,5	3/1	
Эскизный проект	0,1/0,3	1/3	2/2

Приведенные в качестве примера данные (табл. 6) свидетельствуют в частности о том, что при проверке стадии «Формирование требований» степень обнаружения дефектов оказалась меньше нормы на всех стадиях проекта. Обнаружено больше дефектов проектирования непосредственно на той фазе, когда они были произведены, в то время как на более поздних фазах было обнаружено меньше дефектов. Как правило, это достигается посредством проведения инспектирования, под которым понимается проверка всех этапов и стадий проекта (требований, результатов проектирования, программного кода и т. п.) на наличие дефектов.

Инспектирование проводит, как правило, группа коллег автора работы, однако в учебных проектах рекомендуется использовать систему «опеки», когда каждый студент, выполняющий проект, «опекается» своим коллегой и наоборот [55]. Для проведения инспектирования требуется выполнение следующих шагов:

1. Процесс инспектирования начинается с планирования. Планирование включает в себя выбор метрик, по которым будет проводиться инспектирование, а также выбор инструментов для сбора и анализа полученных данных. Результаты проверяемых этапов проекта должны быть логически завершены.
2. Подготовка к инспектированию включает описание классификации дефектов, как правило, на три уровня значимости.
3. Проведение инспектирования в ходе которого инспектирующий заносит все найденные дефекты в базу данных (например, доступную через сеть) вместе с описанием и классификацией.
4. Если дефекты появляются из-за недопонимания или плохого представления, то организуется встреча по анализу причин появления дефектов.
5. На окончательном собрании корректор и автор убеждаются в том, что все дефекты исправлены. Однако это не предполагает детальной ревизии всей работы корректором. Все исправления остаются на совести автора, ответственного за свою работу.
6. Студент-инспектор составляет отчет о проведенной работе, в котором в лаконичной форме (например, в виде таблицы) приводит информацию о выявленных дефектах. Например, информация о найденном дефекте на стадии «Эскизный проект» может быть представлена в следующем виде:

№ дефекта – nn
Стадия – Эскизный проект
Этап – Разработка алгоритма функции поиска
Уровень значимости: высокий
Локализация – наличие дефекта в псевдокоде функции, выполняющей анализ дискриминанта.
FUNCTION D(a,b,c as REAL)
*D=b^2-4*a*c*
IF D > 0
уравнение имеет 2 корня
ELSE уравнение имеет 1 корень
ENDIF
Содержание дефекта:
строка 2: функции присваивается значение дискриминанта, а не результат анализа
строка 3: вариант D<0 и D=0 будет приводить к одному и тому же результату – «уравнение имеет 1 корень». Вместе с тем, при D<0 – корни уравнения комплексно-сопряженные.

Отчет, составленный в ходе инспектирования, подписывается автором работы, инспектирование которой проводилось, и помещается студентом-инспектором в приложение к своей пояснительной записки.

По итогам анализа и планирования качества проектируемой системы разрабатывается план контроля качества программного обеспечения в соответствии со стандартом IEEE 730-1989 (Software Quality Assurance Plan – SQAP), который помещается в приложение ПЗ. Пример составления плана см. в [55]. Оглавление плана контроля качества программного обеспечения приведено в приложении Ж и включает в качестве примера типовое описание некоторых разделов, которое необходимо адаптировать в соответствии с содержанием выполняемого проекта. Примечание к тому или иному разделу приведено в квадратных скобках курсивным шрифтом. Итоговые значения метрик, собранные при выполнении проекта и инспектирования, помещаются в раздел 5.2 SQAP.

4.7.1.5. План управления программным проектом

Управление проектом заключается в управлении производством продукта в рамках отведенных средств и времени. Иными словами, управление проектом – это достижение баланса между

стоимостью, возможностями, качеством и сроками. Возможность управления появляется тогда, когда этот баланс оценивается численно [93].

Существует множество стандартов, описывающих создание и поддержание планов управления программным проектом. В настоящих методических указаниях рекомендуется использовать стандарт IEEE 1058.1-1987 «План управления программным проектом» (Software Project Management Plan – SPMP). План управления программным проектом, оформленный в соответствии с IEEE 1058.1-1987, помещается в приложение ПЗ. Пример составления плана см. в [55].

Принципиальное значение среди всех вопросов, связанных с управлением программным проектом, имеет проблема управления рисками.

Управление рисками включает в свой состав идентификацию, планирование устранения, выбор приоритетов и устранение (или уменьшение влияния).

В разделе «План управления программным проектом» на страницах ПЗ необходимо описать работу по управлению рисками проекта, поскольку дальнейшее проектирование системы должно начинаться с аспектов, имеющих наибольший риск.

Оглавление плана управления программным проектом приведено в приложении И. Оно включает в качестве примера типовое описание некоторых разделов, которое необходимо адаптировать в соответствии с содержанием выполняемого проекта. Примечание к тому или иному разделу приведено в квадратных скобках курсивным шрифтом.

Начиная с раздела ПЗ «План управления программным проектом» в конце каждой главы ПЗ необходимо приводить уточненный план-график (в виде диаграммы Ганта), определяющий, как и когда должны быть выполнены те или иные этапы проекта. Итоговая версия плана-графика помещается в разделе 5.5 SPMP.

4.7.2. Формирование требований к АСОИУ (ИМС)

Все требования к проектируемой системе предлагается размещать на нескольких иерархических уровнях. На самом нижнем уровне располагаются требования, которые одинаково подходят для автоматизированных информационных систем в целом без учета особенностей конкретной прикладной области. Здесь необходимо обратиться к ГОСТам и международным нормативным документам. Далее следует уровень требований к автоматизированной системе определенного (заданного) класса. При этом анализируются соответствующие нормативные документы, определяющие порядок и описание заданного технологического процесса. Например, если на нижнем уровне анализировались требования к автоматизированным системам в целом, то среднему уровню будут соответствовать требования к автоматизированным системам управления технологическим процессом или автоматизированным системам научных исследований. И наконец, третий уровень составляют требования к проектируемой системе.

Существует множество подходов решения задачи системного анализа и документирования требований. В настоящем курсовом проекте для формализации и документирования требований рекомендуется использовать спецификацию требований к программному обеспечению (Software Requirements Specification - SRS) в соответствии со стандартом IEEE 830-1993.

В стандарте IEEE 830-1993 проведено деление степени детализации требований на два уровня. Документы первого уровня (спецификации, диаграммы, эскизы и проч.) описывают потребности заказчика и записывается на языке, понятном заказчику – это т.н. С-требования. На втором уровне детализации С-требования значительно уточняются, т.е. снабжаются максимально возможным количеством деталей. Такой набор документов называют требованиями разработчика, или D-требованиями.

Б**о**льшая часть анализа требований является коммуникационной деятельностью, тщательно организованной для получения наилучших результатов.

4.7.2.1. С- требования

Заказчики разрабатывают концепцию, часто подсознательную и неполную, того, как их приложение будет работать. Эту концепцию иногда называют моделью приложения, или концепцией работы. Для формализации концепции работы приложения, представленной заказчиком, инженеры могут использовать комбинации следующих подходов:

- Методология SADT (SADT – Structured Analysis and Design Technique), представляющая собой набор методов, правил и процедур, предназначенных для построения структурно-

функциональной модели объекта какой-либо предметной области [60, 84, 92] с помощью широкого спектра диаграмм: IDEF0, IDEF3, DFD, ERD и т.д.

- Для выражения взаимодействия приложения с внешним пользователем используют диаграммы вариантов использования (use case) UML (описание нотации UML, как правило, всегда встречается в литературе, посвященной проектированию ИС, например [55, 56, 60, 62, 86, 88, 101, 103]). Варианты использования в последующем могут быть применены для выбора классов, а также для разработки прототипа.
- Прототипирование дизайна пользовательского интерфейса входит в фазу проектирования программного обеспечения, однако его также можно считать и частью фазы формирования требований. Вопросам проектирования и дизайна пользовательского интерфейса посвящена обширная литература. В настоящих МУ рекомендуется воспользоваться подходами, изложенными в [35, 39, 45, 85].

В большинстве случаев при проектировании корпоративных систем для каждого варианта использования составляется комплект из трех диаграмм:

- диаграммы последовательности (sequence diagram) UML отражают коммуникационный аспект реализации варианта использования и включают в свой состав объекты и сообщения между ними;
- диаграммы деятельности (activity diagram) UML отражают управленческий аспект реализации варианта использования и включают в свой состав деятельности, состояния, решения, знаки синхронизации, а также переходы между ними;
- диаграммы классов (class diagram) UML отражают структурный аспект реализации варианта использования и включают в свой состав классы, объекты, экземпляры, а также связи между ними.

Поскольку C-требования предназначены, прежде всего, для заказчиков, диаграммы, используемые для документирования требований на начальной стадии, как правило, имеют упрощенную форму, которая подчеркивает лишь основные концептуальные моменты, а не детали, которые могут затруднять чтение и понимание диаграмм заказчиком.

На стадии формирования C-требований выполняется эскиз пользовательского интерфейса, который представляет собой лишь перечень интерфейсных элементов, достаточных для обеспечения функционального назначения программной системы. Аргументированное расположение интерфейсных элементов на форме, их группировка, выбор размеров, цветовой палитры и т.д. осуществляется на базе концепции ГПИ, представленной в параграфе D-требования.

Как только C-требования собраны, как правило, возникает необходимость обновления SPMP. Такое обновление происходит в течение всего жизненного цикла системы.

После анализа C-требований также может быть уточнена оценка стоимости системы.

Задача формулирования и корректного управления требованиями становится нетривиальной, если количество требований исчисляется несколькими десятками. Вопросам формализации решения этой задачи посвящена обширная литература, англоязычная и русскоязычная части которой достаточно полно представлены в работах [13] и [56, 81, 99], соответственно. Кроме того, разработаны и успешно используются инструментальные CASE-системы управления требованиями (например, RequisitePro, DOORS [99] и др.), которыми рекомендуется воспользоваться в процессе выполнения КП.

4.7.2.2. D-требования

Разработчикам программного обеспечения нужна база для проектирования и разработки. Эта база представляет собой детальные требования (D-требования). D-требования состоят из полного списка конкретных свойств и функциональности, которую должна иметь программа. Каждое из этих требований пронумеровано, помечено и отслеживается по ходу разработки. D-требования должны быть согласованы с C-требованиями.

Существуют несколько типов D-требований:

1. Функциональные требования – определяют работу, которую должна выполнять проектируемая система (например, «приложение будет вычислять стоимость портфеля акций пользователя»).
2. Нефункциональные требования.
   2.1. Производительность. Требования к производительности определяют временные ограничения, которые должны быть выполнены в программе. Заказчики и разработчики обсуждают ограничения по времени вычислений, использованию оперативной памяти, объему запоминающих устройств и т. д.
   2.2. Надежность и безопасность. Требования надежности определяют надежность в измеряемых величинах. Требования такого типа предполагают вероятность неидеальной работы про-

граммы и ограничивают область ее несовершенства. Особое внимание в ПЗ необходимо уделить вопросам обеспечения безопасности системы с учетом возможности искажения данных посредством несанкционированного доступа, сбоя системного или прикладного ПО.

2.3. Обработка ошибок. Эта категория требований объясняет, как программа должна реагировать на возникающие ошибки. Например, что должна делать программа, если она получает сообщение из другой программы в неразрешенном формате? Это не касается ошибок, генерируемых самой программой.

2.4. Интерфейсные требования. Интерфейсные требования описывают формат, в котором программа общается с окружением.

2.5. Ограничения. Ограничения на проектирование или реализацию описывают границы или условия того, как приложение должно быть задумано и разработано. Например, накладываются ограничения на инструменты и языки программирования ввиду сложившихся традиций организации, опыта программистов, совместимости и проч.

3. Обратные требования – это функционал, который система не обеспечивает.

Типичная последовательность получения функциональных D-требований с использованием объектно-ориентированного подхода приведена ниже:

1. Процесс начинается с перечисления классов, упомянутых в вариантах использования – это т.н. классы анализа.
2. Полученный набор классов обычно неполон и следует попытаться найти остальные классы предметной области.
3. Для каждого из полученных классов выписывается вся необходимая функциональность программы, за которую отвечает данный класс. Например, «у каждого клиента будет имя» (атрибут класса Клиент) и «программа должна иметь возможность вычислять капитал каждого клиента» (метод класса Клиент). События, которые должны обрабатывать объекты класса, также должны быть определены. В это же время должны быть разработаны и планы тестирования для каждого D-требования (см. ниже).
4. D-требования проверяются и сравниваются с C-требованиями.
5. D-требования проверяются заказчиком и, затем, публикуются.

Особое внимание необходимо уделить вопросу получения классов анализа, поскольку – это творческий процесс, тесно связанный с анализом информации о предметной области.

При получении классов анализа необходимо учитывать следующие аспекты:

1. Класс анализа является четкой абстракцией, моделирующей один конкретный элемент предметной области. Он должен имеет от 3-х до 5-ти операций. Все свойства класса служат реализации его назначения. Однако для реализации своего назначения класс должен иметь минимальное количество связей с другими классами.
2. В классах анализа содержаться только ключевые атрибуты и операции, определенные на очень высоком уровне абстракции. Как правило, каждая операция уровня анализа затем разбивается на несколько операций уровня проекта.
3. Необходимо избегать глубоких деревьев наследования (3 и более уровней), поскольку частой ошибкой при построении иерархии наследования является использование функциональной декомпозиции, где каждый уровень абстракции имеет только одну операцию. Наследование использует только там, где есть явная иерархия наследования, вытекающая непосредственно из предметной области.
4. При выявлении классов посредством анализа текста C-требования, существительные указывают на классы или атрибуты, а глаголы служат признаком операций. Однако синонимы и омонимы могут стать причиной появления ложных классов.
5. При выявлении классов совместно с пользователями часто используют т.н. CRC(Class-Responsibilities-Collaborators)-анализ с помощью стикеров и мозговой штурм.

Как только все D-требования собраны, необходимо обновить SPMP и передать D-требования под управление конфигурациями.

На протяжении всего процесса проектирования необходимо прилагать усилия по планированию будущего тестирования. Существует несколько преимуществ в написании тестов одновременно с требованиями. Во-первых, это позволяет прояснить требование. Во-вторых, это переносит некоторую часть работы из фазы тестирования проекта в фазу разработки требований. Пусть, например, SRS включает в свой состав D-требование №NNN следующего содержания: «NNN. ФИО каждого студента информационной системы «Деканат» будет представлять собой строку от 1 до 256 симво-

лов». С целью планирования процедуры тестирования необходимо составить проект плана тестирования, примерная форма которого может быть представлена в виде таблицы (см. табл. 7).

Таблица 7

Тестовые данные для проверки D-требования №nnn

№ п/п	Входные тестовые данные	Ожидаемый результат
1	Иванов Иван Иванович	Иванов Иван Иванович
2	length()<10	Сообщение с вопросом о корректности вводимых данных
3	length()>256	
4	…	…

Создание тестовых планов в процессе проектирования системы лежит в основе отдельного направления разработки информационных систем, получившего название Test-Drive-Development (TDD) – разработка, основанная на тестировании. Указанному направлению посвящено множество работ, наиболее известными из которых являются [36, 52].

Результаты анализа требований к проектируемой системе помещаются в спецификацию, оформленную согласно стандарту IEEE 830-1993. Поскольку D-требования предназначены, прежде всего, для разработчиков, то диаграммы, которые составляются для документирования D-требований, должны быть максимально детализированы с использованием возможностей той или иной нотации.

Пример составления спецификации см. в [55]. Оглавление спецификации требований к программному обеспечению приведено в приложении К и включает в качестве примера типовое описание некоторых разделов, которое необходимо адаптировать в соответствии с содержанием выполняемого проекта. Примечание к тому или иному разделу приведено в квадратных скобках курсивным шрифтом.

4.7.2.3. Управление требованиями

Без возможности четкого контроля каждого требования от проекта до программного кода, реализующего это требование, было бы сложно убедиться в том, что программа разработана в соответствии с установленными требованиями. Когда требования меняются (чего следует ожидать), это становится еще сложнее.

Возможность отображать каждое требование на соответствующие части проекта и программы называется прослеживанием.

Один из способов, помогающих обеспечить прослеживание, заключается в отображении каждого функционального D-требования на конкретную функцию целевого языка. На рис. 2 демонстрируется обеспечение принципа прослеживания для некоторого требования №NNN «Поиск учебно-методических материалов (УММ)»: (i) C-требования, сформулированные заказчиком при проведении анкетирования (или интервью), отображаются на диаграмме вариантов использования; (ii) на этапе проработки D-требований, каждое C-требование представляется комплектом из трех диаграмм: деятельности, классов и последовательностей; (iii) при проектировании архитектуры системы для удовлетворения требования №NNN в некотором классе слоя предметной области предусматривается соответствующий метод(ы) (например, «findByAuthor» и «findByTitle)»; (iv) на стадии детального проектирования прорабатывается алгоритм метода, отвечающего за выполнение требования №NNN, а также создается ER-диаграмма БД и конструируется SQL-запрос(ы), выполнение которого обеспечит удовлетворение данного требования; (v) на стадии реализации для выполнения требования №NNN разрабатываются листинги соответствующих методов и, наконец, на стадии тестирования выполнение требования №NNN проверяется с использованием составленного плана тестирования.

При формировании требований к АСОИУ необходимо обеспечить достижение ряда показателей, среди которых наиболее значимыми являются: прослеживание, полнота и согласованность. Набор D-требований согласован, если между требованиями нет противоречий. По мере увеличения числа D-требований согласованность может стать труднодостижимой, но объектно-ориентированная организация требований помогает избежать несогласованности благодаря классификации D-требований по классам и с помощью разложения их на простейшие составляющие.

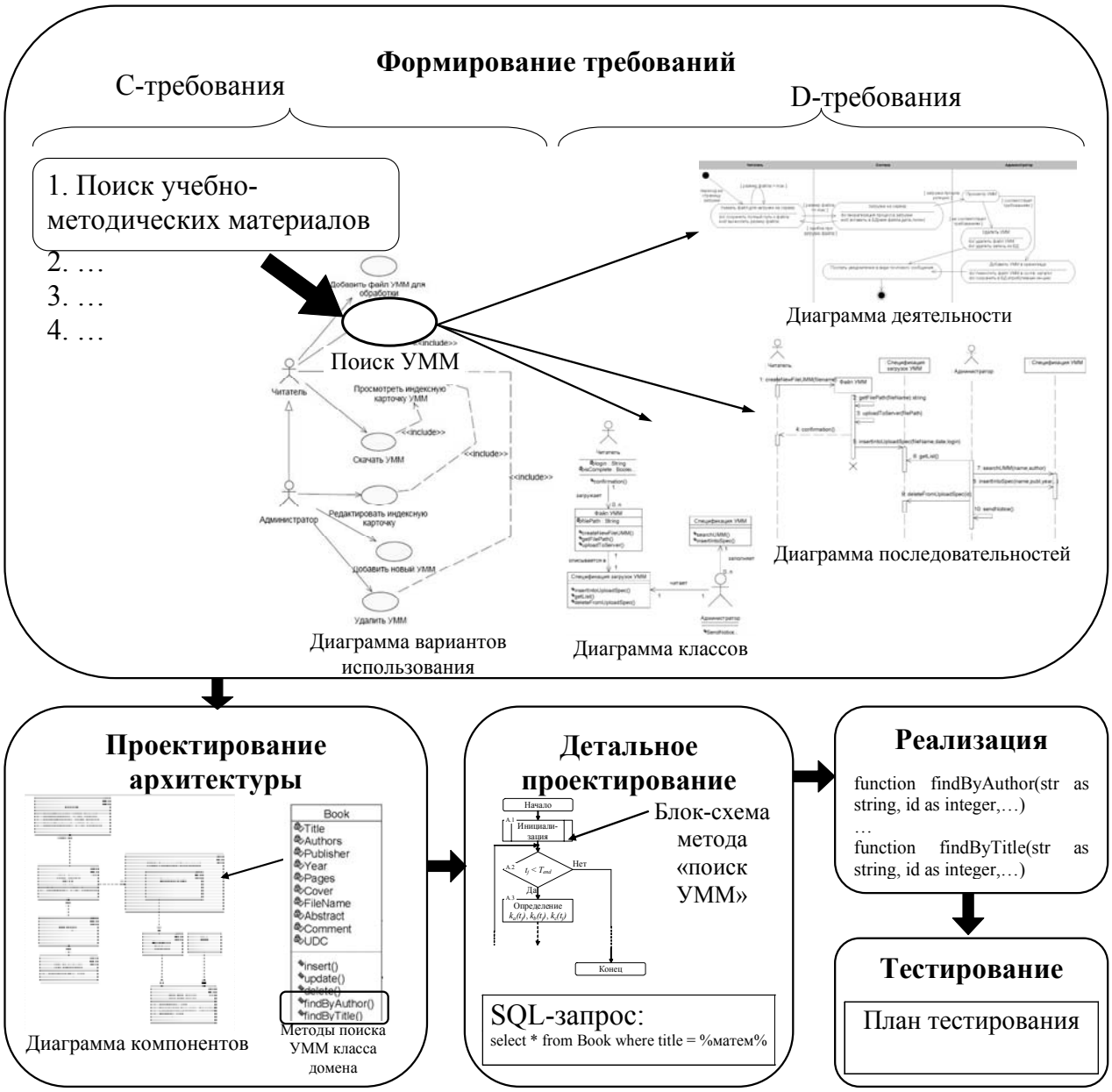

Рис. 2. Пояснение принципа прослеживания требований

При выполнении КП необходимо особое внимание уделить вопросам прослеживания требований. С этой целью в тексте и на диаграммах ПЗ делают отметки, позволяющие локализовать положение каждого С-требования в проектируемой системе.

С целью проверки указанных характеристик составляют таблицу, форма которой приведена на рис. 3.

Во втором столбце таблицы приводится идентификатор D-требования. В столбце 3 приводится идентификатор С-требования, в состав которого входит данное D-требование. В столбце 4 приводятся идентификаторы D-требований, выполнение которых необходимо и достаточно для реализации данного D-требования. В столбце 5 делают отметку, если данное D-требование не противоречит никакому другому D-требованию из перечисленных в таблице. В столбце 6 делают отметку, если данное D-требование может быть реализовано с учетом ограничений, накладываемых другими С- или D-требованиями, средой разработки, аппаратной платформой и т.д.

№ п/п	Требование	Прослеживание назад	Полнота	Согласованность	Выполнимость	Однозначность	Ясность	Точность	Модифицируемость	Тестируемость	Прослеживание вперед
1	2	3	4	5	6	7	8	9	10	11	12

Рис. 3. Шаблон таблицы для проверки D-требований

В столбце 7 делают отметку, если данное D-требование не может быть истолковано двояко. В столбце 8 делают отметку, если данное D-требование сформулировано предельно просто. В случае отсутствия отметки необходимо привести дополнительные разъяснения по содержанию такого требования. В столбце 9 делают отметку, если в данном требовании указаны допустимые отклонения от необходимого результата. В столбце 10 делают отметку, если формулировка данного D-требования не претерпит существенных изменений при возникновении в будущем необходимости наращивания функциональных возможностей системы. В столбце 11 делают отметку, если для данного требования составлен план тестирования. В столбце 12 делают отметку по окончании реализации данного D-требования в виде программного кода.

В заключение раздела, посвященного вопросам управления требованиями, необходимо еще раз подчеркнуть сущность принципа документирования артефактов[1], принятого в настоящих МУ. Большинство артефактов, которые создаются на протяжении выполнения проекта, проходят две стадии детализации: при первом упоминании на страницах ПЗ приводится черновой эскиз (или сжатое предварительное описание), а затем прорабатывается итоговый вариант (рис. 4). Этот принцип, прежде всего, относится к диаграммам, которые создаются при формировании требований во 2-ой главе ПЗ: черновые эскизы диаграмм и их описания приводятся на страницах ПЗ, итоговые (полные) версии помещаются в приложение (SRS). Необходимо отметить, что именно приложения к ПЗ представляют собой тот комплект конструкторско-технологической документации, который и соответствует понятию «проект», поскольку в приложения помещают, представляя в лаконичной, законченной форме результаты, полученные на страницах ПЗ.

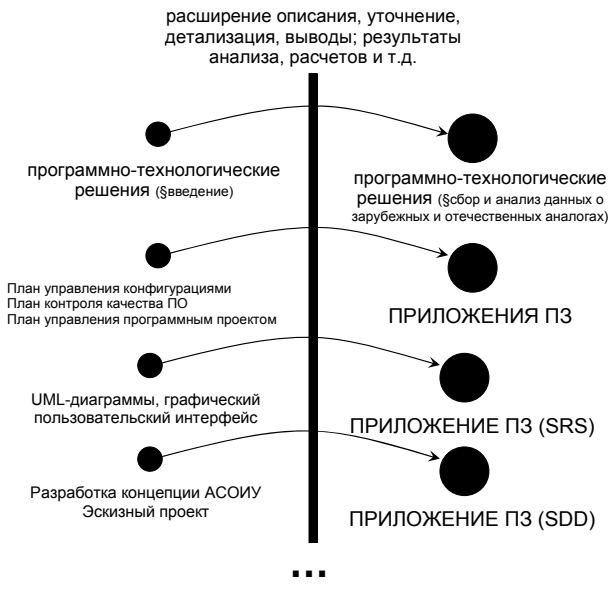

Рис. 4. Пояснение принципа документирования артефактов ПЗ

Как будет показано в следующем разделе, эскиз графического пользовательского интерфейса также прорабатывается в два этапа.

4.7.2.4. Проектирование пользовательского интерфейса

Проектирование графического пользовательского интерфейса (ГПИ) в настоящий момент представляет собой междисциплинарное направление, включающее в свой состав аспекты дизайна, анализа требований, системного проектирования, программирования, эргономики, социальной психологии и многие другие вопросы. Переход от традиционного ГПИ к объектно-ориентированному пользовательскому интерфейсу, развитие концепций социализированных пользовательских интерфейсов, адаптивных интерфейсов с мультиагентной поддержкой, интерфейсов, построенных с использованием элементов искусственного интеллекта, требует особых усилий при проектировании

[1] Под артефактами (в терминологии USDP) подразумеваются определения, требования, диаграммы, листинги кодов, планы и т.д., т.е. все то, что является продуктом той или иной стадии создания ИС. Как правило, эволюцию артефактов необходимо отслеживать с помощью системы управления конфигурациями.

пользовательского интерфейса, отвечающего современному уровню развития человеко-машинного взаимодействия.

Ввиду ограниченного объема времени, выделенного на курсовое проектирование, проработку ГПИ следует выполнять в два этапа: (i) подготовка чернового эскиза пользовательского интерфейса, представляющего собой перечень интерфейсных элементов, необходимых для обеспечения требуемой функциональности; (ii) подготовка итогового эскиза путем переработки чернового варианта ГПИ в соответствии с описанной на страницах ПЗ концепцией.

Концепция ГПИ, представленная на страницах ПЗ, должна основываться на классических и современных подходах к разработке и дизайну ГПИ [35, 39, 45, 85] и устанавливать требования к:

- цветовым схемам, геометрическим формам, визуальным закономерностям, пропорциям, контрасту расположения элементов, принятых во всем интерфейсе в целом;
- подсказкам, пояснениям, сообщениям об ошибках, справочной информации в соответствии с особенностями пользователей системы, составляющих наиболее многочисленную группу;
- отдельным элементам, используемым при построении ГПИ (кнопки, чекбоксы и радиокнопки, списки, поля ввода, счетчики и ползунки, меню и т.д.), в соответствии с их каноническим назначением и порядком применения;
- защите системы от ошибок пользователей через проработку блокировок потенциально опасных действий, проверку всех действий пользователя перед их принятием, обеспечение возможности отмены пользователем своих предыдущих действий, без ограничения количества уровней отмены и типа отменяемых действий и проч.

Известны несколько фундаментальных принципов, относящихся ко многим элементам интерфейса:

1. Правило, сформулированное в 1954 году Полем Фитсом: «Время достижения цели прямо пропорционально дистанции до цели и обратно пропорционально размеру цели». Откуда, в частности, следует, что, например, повысить скорость работы можно посредством увеличения площади активного элемента, что бы пользователь меньше времени тратил на позиционирование курсора мыши.
2. Принцип «релевантное – вперед»: чаще всего используемые элементы должны быть расположены в левой верхней части экрана, реже используемые – в правой нижней части.
3. При группировке элементов меню или, например, списков необходимо помнить ограничение, накладываемое кратковременной памятью человека: запоминаются 7±2 пункта.
4. При размещении графических элементов пользовательского интерфейса на формах (или страницах) настоятельно рекомендуется использовать модульную сетку.
5. Все размеры и координаты (как минимум пропорции диалоговых окон) необходимо привязывать к золотому сечению (0.618 x 0.382).

В главе 2 на страницах ПЗ формулируется концепция ГПИ в объеме, достаточном для проработки чернового эскиза интерфейса. И черновой и итоговый эскизы ГПИ помещаются в разделы SRS 2.1.2 и 3.1.1 соответственно.

Необходимо обратить внимание на то, чтобы каждый шаг преобразования чернового эскиза в итоговый был обоснован и снабжался ссылкой на соответствующий пункт концепции.

4.8. РАЗРАБОТКА КОНЦЕПЦИИ АСОИУ (ИМС)

Раздел, посвященный разработке концепции АСОИУ, играет ключевую роль в курсовом проекте, поскольку посвящен созданию архитектуры системы.

Описанию различных архитектур программного обеспечения посвящен широкий спектр литературы [10, 14, 15, 34] и в настоящее время принципы построения эффективных архитектурных решений ИС составляют отдельное бурно развивающееся направление информационных технологий.

В настоящих методических указаниях используется классификация, предложенная Шоу и Гарланом [55], согласно которой ИС можно разделить на следующие классы:

1. Потоки данных – архитектура, в которой независимо спроектированные и функционирующие модули обрабатывают потоки данных, перемещающиеся от одного модуля к другом. Представление архитектуры информационной системы в виде диаграмм потоков данных является универсальным методом описания подавляющего большинства приложений, однако при отображении диаграмм на программный код может возникнуть неопределенность.
2. Архитектура независимых компонентов состоит из компонентов, функционирующих параллельно во времени.

2.1. Наиболее ярким примером такого типа архитектуры является клиент-серверная. Существуют многочисленные модификации клиент-серверной архитектуры, в частности, широкое распространение получили трехуровневые системы, в которых помимо традиционных клиента и сервера выделен промежуточный уровень, отвечающий за преобразование данных и их перенаправление (например, CORBA, COM).

2.2. Архитектура параллельно взаимодействующих процессов характеризуется тем, что в ней одновременно запускается несколько процессов (в различных потоках).

2.3. Приложение, построенное по архитектуре событийно-управляемых систем, представляет собой набор компонентов, каждый из которых находится в состоянии ожидания, пока не произойдет воздействующее на него событие.

3. Архитектура виртуальных машин рассматривает приложение как программу, написанную на специальном языке. При этом, поскольку должен быть реализован интерпретатор этого языка, использование такой архитектуры окупится, если будут реализованы несколько программ.
4. Большинство систем, построенных в соответствии с репозиторной архитектурой предназначены для обработки транзакций по отношению к БД. Примеры архитектур подобного класса можно найти в области итеративных сред разработки, которые позволяют редактировать и компилировать в БД коды и объектные файлы.

Для конкретного проекта ИС может быть найдено несколько подходящих архитектур, из которых необходимо выбрать лучшую.

Для выбора архитектуры ИС необходимо на страницах ПЗ провести анализ существующих типов архитектур [10, 14, 15, 34], ввести систему критериев и осуществить выбор архитектуры, воспользовавшись методикой, описанной в п. 4.7.1.3.

Проектирование архитектуры тесно связано с понятием модуля (подсистемы), представляющего собой фрагмент программного кода, и состоящего из интерфейсной части и части реализации.

Модульность – свойство системы, при которой система может подвергаться декомпозиции на ряд внутренне связанных и слабо зависящих друг от друга частей.

Целями декомпозиции на подсистемы являются:

- повышение уровня абстракции системы;
- декомпозиция системы на части, которые могут независимо конфигурироваться или поставляться, разрабатываться (при условии стабильности интерфейсов), размещаться в распределенной среде, изменяться без воздействия на остальные части системы;
- разделение системы на части из соображений ограничения доступа к основным ресурсам;
- представление существующих продуктов или внешних систем (компонентов), которые не подлежат изменениям.

Ключевая задача проектирования архитектуры системы состоит в соблюдении двух принципов [62, 80, 89, 98]:

- принцип «слабой вешней связанности» (**Low Coupling**) – количество связей между отдельными подсистемами должно быть минимальным;
- принцип «сильного внутреннего сцепления» (**High Cohesion**) – связность отдельных частей внутри каждой подсистемы должна быть максимальной.

Вопросам проектирования и реализации корпоративных информационных систем в настоящее время посвящено значительное количество работ [16, 48, 49] и, более того, активно внедряются и используются различного рода платформы (ERP-системы, инструментальные среды разработки (IDE), системы управления содержимым (CMS) и т.д.), которые позволяют в рамках единой программно-алгоритмической среды реализовать специфические для заданной предметной области требования. В этой связи, в большинстве случаев, задача состоит не в создании информационной системы в целом или ее части с «чистого листа», а в отыскании и адаптации готовых решений, эффективность которых неоднократно проверена на практике. Такие программно-алгоритмические решения, получив название шаблонов (паттернов) проектирования, широко применяются на практике при проектировании архитектуры ИС [62, 80, 98].

На первом этапе проектирование архитектуры выполняется на уровне классов системы и преимущественно состоит в распределении обязанностей между классами с использованием паттернов GRASP (General Responsibility Assignment Software Patterns). Из числа GRASP-паттернов наибольшее распространение получили [80]:

- **информационный эксперт (Information Expert)** – правило, позволяющее выбрать класс (из нескольких кандидатур) для назначения тех или иных функциональных обязанностей;

- **создатель (Creator)** – правило, позволяющее назначить классу B обязанность создавать экземпляры класса A при выполнении определенных условий;
- **контроллер (Controller)** – создание специального (служебного) класса–обработчика системных сообщений.

Кроме перечисленных выше пяти базовых шаблонов, отвечающих за распределение обязанностей между классами, дополнительно используются шаблоны GRASP, предназначенные для обеспечения универсальности ИС и/или расширяемости ее в будущем:

- **полиморфизм (Polymorphism)** – порядок распределения обязанностей класса для обеспечения различных вариантов поведения его объектов;
- **чистая синтетика (Pure Fabrication)** - правила создания служебного класса, обеспечивающего высокое сцепление, низкое связывание и повторное использование;
- **перенаправление (Indirection)** – правила создания промежуточного (служебного) класса, предназначенного для обеспечения связи между компонентами или службами системы, связывание которых напрямую привело бы к нежелательному повышению внешней связности;
- **защищенные вариации (Protect Variation)** – правила проектирования подсистемы, при выполнении которых изменение ее объектов не оказывает существенного влияния на другие подсистемы.

Последний из указанных выше паттернов в большей степени отражает проблемы проектирования интерфейсов системы, решению которых посвящены многие **GoF-паттерны** (Gang-of-Four – союз четырех), опубликованные в [62].

Использование паттернов позволяет решить несколько фундаментальных задач проектирования информационных систем – обеспечить удовлетворение принципов «слабой вешней связанности и сильного внутреннего сцепления», а также обеспечить возможность повторного использования компонентов. Ниже перечислены ошибки проектирования, затрудняющие повторное использование компонентов, а также наращивание функциональности системы и **GoF-паттерны**, позволяющие исправить такие ошибки [62]:

- поскольку сильно связанные между собой классы образуют монолитные подсистемы, в которых нельзя ни изменить, ни удалить класс без модификации других классов. Для снижения внешней связности используются слои (см. ниже), абстрактные связи, а также следующие паттерны: **абстрактная фабрика (abstract factory)**, **мост (bridge)**, **цепочка обязанностей (chain of responsibility)**, **команда (command)**, **фасад (facade)**, **посредник (mediator)**, **наблюдатель (observer)**. Для повышения независимости отдельных классов подсистемы друг от друга используются паттерны: **адаптер (adapter)**, **декоратор (decorator)**, **посетитель (visitor)**.
- использование имени класса при создании объекта привязывает проектировщика к конкретной реализации, а не к интерфейсу, что может осложнить изменение объектов в будущем. Решение этой проблемы состоит в косвенном создании объектов с помощью паттернов **абстрактная фабрика**, **фабричный метод (factory method)**, **прототип (prototype)**;
- использование конкретной операции ограничивает выполнение запроса одним единственным способом. Паттерны **цепочка обязанностей** и **команда** позволяют, не включая запросы в код, изменить способ удовлетворения запроса, как на этапе компиляции, так и на этапе выполнения;
- ограничить зависимость от конкретной платформы (операционной системы с предоставляемой ее API) позволяют паттерны **абстрактная фабрика** и **мост**;
- если при построении клиента использовалась информация о том, как тот или иной объект представлен, храниться или реализован, то при изменении объекта может потребоваться изменить клиент. Инкапсуляцию перечисленной информации обеспечивают паттерны **абстрактная фабрика**, **мост**, **хранитель (memento)**, **заместитель (proxy)**;
- объекты, зависящие от изменяющихся алгоритмов (в случае необходимости расширения, оптимизации и т.д.), приходится перепроектировать. Паттерны **мост**, **итератор (iterator)**, **стратегия (strategy)**, **шаблонный метод (template method)** и **посетитель** позволяют изолировать алгоритмы с высокой вероятностью изменения;
- композиция объектов и делегирование – гибкие альтернативы наследования для комбинирования поведения, поскольку в этом случае новая функциональность добавляется в систему посредством изменения способа композиции объектов, а не за счет определения новых подклассов через наследование. Однако использование только композиции объектов может су-

щественно усложнить структуру проекта. В этой связи используют прием, при котором вначале определяется подкласс, а затем для обеспечения требуемой специализации выполняется комбинирование его объектов с существующими. Такой прием нашел отражение в следующих паттернах: **мост**, **цепочка обязанностей**, **компоновщик (composite)**, **декоратор (decorator)**, **наблюдатель**, **стратегия**.

Определив обязанности классов, введя системные класса, а также, рассмотрев интерфейсы классов в соответствии с паттернами GRASP и GoF, переходят к описанию более высокоуровневых абстракций – архитектурных слоев ИС.

Архитектура большинства корпоративных приложений включает в свой состав три слоя [98]: (i) слой представления (presentation) предназначен для обработки событий пользовательского интерфейса, отображении данных, поддержки функций командной строки и т.д.; (ii) слой предметной области (бизнес-логики или домена) предназначен для описания бизнес-логики – основных функций приложения, предназначенных для достижения поставленной цели; (iii) слой источника данных (data source) – отвечает за обращение к БД, обмен сообщениями, управление транзакциями с внешними приложениями.

Зависимость слоев бизнес-логики и источника данных от слоя представления категорически не допускается – это позволяет упростить адаптацию слоя представления или замену его альтернативным вариантом с сохранением основы приложения.

4.8.1. Слой предметной области

При решении вопроса физического размещения слоев в первом приближении стараются обеспечить их функционирование на сервере, что существенно упрощает процедуры исправления ошибок и обновления версий. Однако в этом случае снижается скорость реагирования и возникает необходимость постоянной поддержки сетевого соединения. Если для проектируемой ИС указанные характеристики являются критическими, то слой представления и слой бизнес-логики вынуждены размещать на клиентской стороне. Однако, расщепление слоя бизнес-логики между клиентом и сервером стараются избегать.

Наибольшее распространение получили три подхода к организации **слоя бизнес-логики**:

1. Типовое решение «**сценарий транзакции**» **(transaction script)** предполагает использование процедуры, которая получает на вход информацию от слоя представления, обрабатывает ее, проводя необходимые проверки и вычисления, при необходимости активирует операции других систем. Затем процедура возвращает результат работы слою представления, нужным образом форматируя его, и при необходимости передает результат работы слою источника данных. Таким образом, в этом случае бизнес-логика представляет набор процедур, по одной на каждую операцию (или вариант использования), которые могут храниться в различных подпрограммах.

   Существует два способа разнесения кода сценариев транзакции по классам: (i) использование одного класса для реализации нескольких сценариев транзакции; (ii) разработка собственного класса для каждого сценария транзакции. При этом создается родительский класс с базовым методом, который наследуется классом, реализующим выполнение заданного сценария.

   Достоинства типового решения «сценарий транзакций»: простота и наглядность, а также четкие границы транзакции. Недостаток – существенное усложнение структуры приложения и дублирование кода при усложнении бизнес-логики.
2. Типовое решение «**модель предметной области**» **(domain model)** строится на базе объектно-ориентированного подхода. Вместо использования одной подпрограммы, несущей в себе всю логику, которая соответствует набору функций, реализуемых системой, каждый объект наделяется только теми методами, которые отвечают его природе.

   В первом приближении модели предметной области делят на два типа: (i) «простые» модели предметной области. В этом случае структура домена совпадает с ER-моделью базы данных: одному объекту домена соответствует одна таблица БД; (ii) «сложные» модели – структура домена не совпадает с ER-моделью БД и включает в свой состав иерархии наследования, а также сложные сети мелких взаимосвязанных объектов.

   Применение модели предметной области для реализации бизнес-логики ИС сопряжено с дополнительными трудностями, связанными с отображением объектов в реляционные структуры БД. Однако существует несколько способов решения этой задачи, одним из которых является использование **преобразователя данных** (см. далее) в качестве паттерна слоя источника данных.

Рассмотрим для примера организацию программной системы, предназначенной для формирования сводного отчета по успеваемости студентов. Диаграмма последовательностей функционирования системы, построенной на базе паттерна «сценарий транзакций», приведена на рис. 5.

Анализ диаграммы, представленной на рис. 5, показывает, что все обязанности по формированию отчета сосредоточены в объекте «*Служба формирования отчетов*». В том случае, если метод «вычислитьРезультат» является тривиальным (например, когда необходимо подсчитать лишь количество студентов первого курса, сдавших сессию на «4» и «5»), то использование сценария транзакций является вполне обоснованным. Однако зачастую бизнес-логика предметной области, в особенности корпоративных приложений, может включать десятки и даже сотни сложно-структурированных правил, что существенно затрудняет разработку и последующую модификацию объектов слоя домена.

Рис. 5. Формирование отчета в системе, построенной на базе паттерна «**сценарий транзакций**»

При использовании паттерна «**модель предметной области**» (см. рис. 6) обязанности каждого объекта строго разграничиваются и при необходимости наращивания функциональных возможностей системы изменениям будут подвержены отдельные методы, в то время как структура домена будет оставаться неизменной.

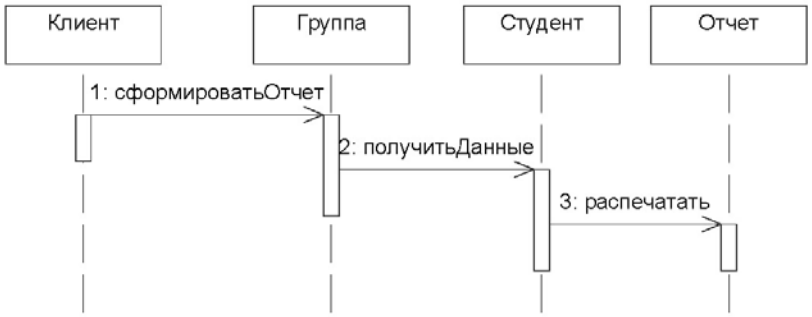

Рис. 6. Формирование отчета в системе, построенной на базе паттерна «**модель предметной области**»

3. Типовое решение «**модуль таблицы**» **(table module)** является промежуточным вариантом между сценарием транзакций и моделью предметной области. Модуль таблицы основан на использовании паттерна источника данных «**множество записей**». Большинство инструментальных платформ (например, Microsoft COM и .NET) позволяют работать с результатами обработки SQL-запроса, организованными в виде множества записей. При этом имеется возможность выполнения запроса, манипулирования его результатом в контексте модуля таблицы и передачи данных графическому интерфейсу для отображения.

   Типовое решение «модуль таблицы» предусматривает создание по одному классу на каждую таблицу БД, единственный экземпляр класса содержит всю логику обработки данных таблицы.

Основное отличие паттернов «модуль таблицы» и «модель предметной области» состоит в том, что если, например, приложение обслуживает множество заказов, в соответствии с моделью предметной области необходимо создать по одному объекту на каждый заказ, а при использовании модуля таблицы понадобится всего один объект, представляющий одновременно все заказы.

В большинстве случаев для решения общей задачи в процессе функционирования системы создается несколько модулей таблицы, которые могут манипулировать одним и тем же множеством записей.

Вместе с тем, модуль таблицы не позволяет воспользоваться всеми возможностями объектно-ориентированной методологии для реализации сложной бизнес-логики (в частности, не полноценно функционирует механизм полиморфизма). В этой связи, реализация крупных корпоративных приложений возможна только с использованием **модели предметной области**, не имеющей ограничений.

Нередко при реализации слоя бизнес-логики его расщепляют на два подуровня, выделяя помимо модели предметной области или модуля таблицы, **слой служб**, который предназначен, например, для управления транзакциями или обеспечения безопасности. Кроме того, рассмотренные типовые решения реализации слоя бизнес-логики не являются взаимоисключающими. Например, сценарий транзакций может использоваться для некоторого фрагмента бизнес-логики, а модель предметной области или модуль таблицы – для оставшейся части.

4.8.2. Слой источника данных

Роль слоя источника данных в основном состоит во взаимодействии с БД в большинстве случаев реляционной. Получили распространение следующие подходы к реализации слоя источника данных:

1. Для поддержания хорошей переносимости код SQL стремятся обособлять от бизнес-логики, размещая его в специальных классах, в которых воспроизводится структура каждой таблицы БД и формируется шлюз, поддерживающий возможность обращения к таблице. При этом используется два варианта реализации типового решения «**шлюз**» (**gateway**):
   a. **шлюз записи данных (row data gateway)** предполагает использование экземпляра шлюза для каждой записи, возвращаемой в результате обработки запроса к БД. При такой реализации шлюз записи данных выступает в роли объекта, полностью повторяющего одну запись, например, одну строку таблицы БД (каждому столбцу таблицы соответствует поле записи).

      При реализации шлюза записи данных часто возникают трудности в принятии решения о размещении метода поиска. В том случае, если необходимо взаимодействовать с разными источниками данных рекомендуется для каждой таблицы БД создавать отдельный класс для проведения поиска и отдельный класс для сохранения результатов, как показано на рис. 7.

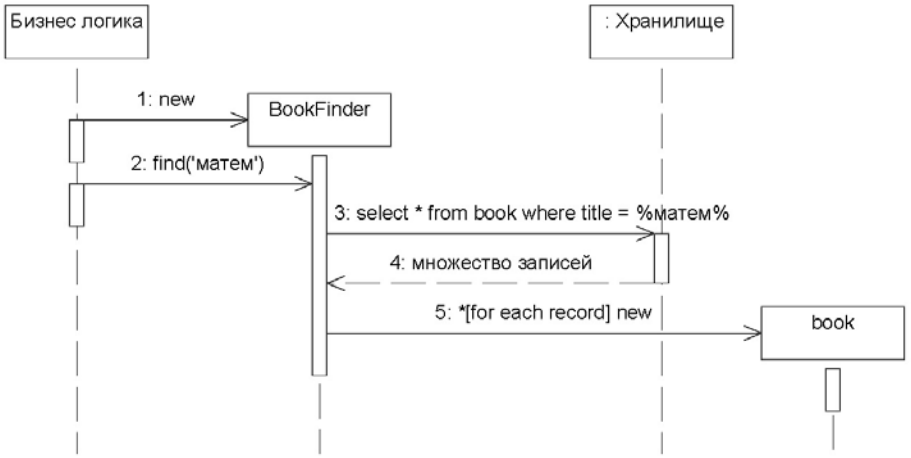

Рис. 7. Реализация метода поиска в контексте **шлюза записи данных**

      Как правило, шлюз записи данных используется при представлении бизнес-логики в виде сценария транзакций.

   b. **шлюз таблицы данных (table data gateway)** предполагает использование классов, сопоставленных с отдельной таблицей БД, и содержащих методы активации запросов, возвращающие множество записей. Модель «**множества записей**» – основополагающая структура данных,

имитирующая табличную форму представления содержимого БД. Различными инструментальными системами (IDE) поддерживаются графические интерфейсные элементы, реализующие паттерн «**множества записей**». Интерфейс **шлюза таблицы данных** включает в себя несколько методов поиска, предназначенных для извлечения данных, а также методы обновления, вставки и удаления. Каждый метод передает аргументы соответствующей команде SQL и выполняет ее в контексте установленного соединения с БД.

2. При описании бизнес-логики с использованием модели предметной области, относящейся к типу «простой», для представления слоя источника данных может использоваться типовое решение «**активная запись**» **(active record)**. Каждая активная запись отвечает за сохранение и загрузку информации в БД, а также за логику домена, применяемую к данным. Структура данных **активной записи** должна в точности соответствовать структуре таблицы БД: каждое поле объекта должно соответствовать одному столбцу таблицы. Как правило, **активная запись** включает в себя методы, предназначенные для выполнения следующих операций: (i) создание экземпляра активной записи на основе строки, полученной в результате выполнения SQL-запроса; (ii) создание нового экземпляра активной записи для последующей вставки в таблицу; (iii) статические методы поиска, выполняющие стандартные SQL-запросы и возвращающие активные записи; (iv) обновление БД и вставка в нее данных из активной записи; (v) извлечение и установка значений полей (get- и set-методы); (vi) реализация фрагментов бизнес-логики. Недостатком активной записи является тесная зависимость структуры ее объектов от структуры БД.
3. При использовании «сложной» модели предметной области для организации слоя бизнес-логики стремятся полностью изолировать модель предметной области от БД, возложив на слой источника данных всю ответственность за отображение объектов домена в таблицы БД. В этом случае слой источника данных обслуживает все операции загрузки и сохранения информации, инициируемые бизнес-логикой и позволяет независимо модифицировать как модель предметной области, так и схему БД. Поскольку объекты и реляционные СУБД используют разные механизмы структурирования данных, то в реляционных БД не отображаются многие характеристики объектов. В этой связи, для обеспечения обмена данными между объектной моделью и реляционной СУБД используют паттерн «**преобразователь данных**» **(data mapper)** – слой ПО, который отделяет объекты, расположенные в оперативной памяти от БД. В функции преобразователя данных входит: передача данных между объектами и БД и изоляция их друг от друга; обработка классов, поля которых объединены одной таблицей; обработка классов, соответствующих нескольким таблицам, классов с наследованием, а также обеспечение связывания загруженных объектов.

Реализация перечисленных выше функций является нетривиальной задачей, в состав преобразователя данных может входить ряд узкоспециализированных паттернов, которые, однако, находят широкое применение при проектировании слоя источника данных в целом. Из числа «служебных» типовых решений источника данных наибольшее распространение получили следующие паттерны:

1. **единица работы (unit of work)** - содержит список объектов, охватываемых бизнес-транзакцией и предназначен для управления изменениями в БД и решения проблемы параллелизма;
2. **коллекция объектов (identity map)** – гарантирует, что каждый объект будет загружен из БД только один раз, сохраняя загруженный объект в специальной коллекции. При получении запроса коллекция просматривается в поисках нужного объекта;
3. **загрузка по требованию (lazy load) –** представляет собой объект, который не сдержит всех требующихся данных в настоящий момент времени, но может их загрузить в случае необходимости;
4. **поле идентификации (identity field)** – сохраняет идентификатор записи данных для поддержки соответствия между объектом приложения и строкой БД;
5. **отображение внешних ключей (foreign key mapping) –** паттерн, предназначенный для отображения ассоциации между объектами на ссылки внешнего ключа между таблицами БД;
6. **отображение с помощью таблицы ассоциации (association table mapping) –** сохраняет множество ассоциаций в виде таблицы, содержащей внешние ключи таблиц, связанных ассоциациями;
7. **отображение зависимых объектов (dependent mapping) –** передает полномочия некоторому служебному классу по выполнению отображения для дочернего класса;
8. **внедренное значение (embedded value) –** отображает объект на несколько полей таблицы, соответствующей другому объекту;
9. **сериализованный крупный объект (serialized large object)** – сохраняет граф объектов путем их сериализации в единый крупный объект и помещает его в поле БД;

10. **наследование с одной таблицей (single table inheritance) –** представляет иерархию наследования классов в виде одной таблицы, столбцы которой соответствуют всем полям классов, входящим в иерархию;
11. **наследование с таблицами для каждого класса (class table inheritance)** – представляет иерархию наследования классов, используя по одной таблице для каждого класса;
12. **наследование с таблицами для каждого конкретного класса (concrete table inheritance)** – представляет иерархию наследования классов, используя по одной таблице для каждого конкретного класса этой иерархии;
13. **преобразователи наследования (inheritance mappers)** – структура, предназначенная для организации преобразователей, которые работают с иерархиями наследования.

4.8.3. Слой представления

Приступая к проектированию слоя представления ИС, в первую очередь, делается выбор типа интерфейса между «толстым» и «тонким» клиентами. Толстый клиент позволяет обеспечить богатый графический интерфейс и высокую скорость отклика системы, однако, требует дополнительных расходов по внедрению и сопровождению клиентской части системы. В этой связи, на первом этапе проектирования слоя представления рассматривается возможность использования тонкого клиента и только в случае неудовлетворения требований обращаются к модели толстого клиента. Как показывает практика, при разработке корпоративных приложений возможностей тонкого клиента оказывается вполне достаточно для реализации всего комплекса требуемых функций. В этой связи ниже будут рассматриваться типовые паттерны слоя представления тонкого клиента.

Для разработки web-приложений наибольшее распространение получил шаблон «**модель-представление-контроллер**» (**Model View Controller - MVC**) (см. рис. Л.1). Контроллер принимает запрос, извлекает из него информацию и передает управление соответствующему объекту **модели**, которая может быть представлена, например, в виде **модели предметной области** или **сценария транзакции**. **Модель** обращается к источнику данных и выполняет действия, предусмотренные в запросе. Затем управление возвращается **контроллеру**, который, анализируя полученный результат, принимает решение о выборе варианта представления ответа и передает управление слою **представления**. Слой представления отображает результат обработки средствами графического интерфейса. Как правило, взаимодействие контроллера и представления осуществляется не напрямую, а через тот или иной объект HTTP-сеанса.

В большинстве случаев, для одной и той же **модели** создается несколько **представлений**, однако при внесении изменений в **модель** эти изменения должны быть отражены во всех представлениях, использующих модель. Решение этой задачи достигается за счет использования типовых решений **наблюдатель (observer)** или **слушатель (listener)**[2] [62].

Для реализации представления в модели MVC используются три типовых решения:

1. **представление по шаблону (template view)** соответствует размещению в структуре web-страницы специальных маркеров, указывающих на расположение динамического содержимого (см. рис. 8).

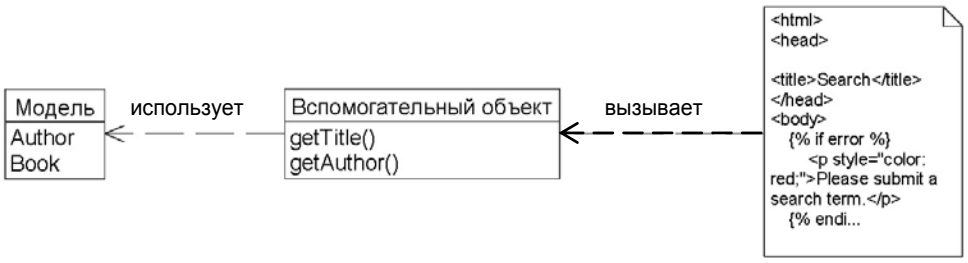

Рис. 8. Пояснение структуры паттерна «**представление по шаблону**»

Многие инструментальные платформы (ASP, JSP, PHP) позволяют в текст web-страницы внедрять программный код (скриптлет), написанный на том или ином языке сценариев. Однако использование скриптлетов приводит к смешиванию слоев программной системы, вызывает дубли-

[2] При использовании типовых решений «**наблюдатель**» или «**слушатель**» представление отслеживает изменение модели и как только изменение происходит генерирует событие по которому все остальные представления обновляют свое содержимое.

рование кода и затрудняет применение многих средств структурирования, поддерживаемых как объектно-ориентированной, так и структурно-функциональной методологиями. В этой связи использование **представления по шаблону** предполагает назначение каждой странице вспомогательного объекта (helper object), который содержит в себе всю логику домена. При этом страница лишь выполняет вызовы вспомогательного объекта.

Строгое разделение задач оформления страницы и проработки программной логики при использовании представления по шаблону позволяет добиться полной независимости работы графических дизайнеров и программистов. Недостаток типового решения **представление по шаблону** связан с тем, что его использование в большинстве случаев возможно исключительно совместно с web-сервером, что затрудняет проведение тестирования. Этот недостаток исключен в типовом решении «**представление с преобразованием**», которое может функционировать при отсутствии запущенного web-сервера.

2. **представление с преобразованием (transform view)** предполагает использование специальной программы (преобразователя), которая просматривает данные домена и преобразует их в код HTML (рис. 9).

Рис. 9. Пояснение структуры паттерна «**представление с преобразованием**»

В настоящий момент наиболее популярным языком для написания представлений с преобразованием является функциональный язык программирования XSLT. При этом данные домена должны храниться в формате XML. Использование XSLT оказывается наиболее эффективным, когда требуются частые (и глобальные) изменения внешнего вида web-сайта.

3. **двухэтапное преобразование (two step view)** предполагает использование двух стадий: вначале на основе данных домена формируется логическая структура (включая поля ввода, верхние и нижние колонтитулы, таблицы, переключатели и т.д.), которая затем трансформируется в код HTML. Методам второго этапа известно, какие элементы входят в логическую структуру и как визуализировать каждый из них. В этом случае, при необходимости, глобальные изменения внешнего вида Web-сайта могут быть осуществлены только переработкой второго этапа преобразования, что повышает эффективность данного типового решения с точки зрения удобства переработки графического интерфейса.

В большинстве случаев двухэтапное преобразование используется для обеспечения доступа к web-контенту с помощью разнообразных обозревателей или когда необходимо поддерживать нескольких вариантов внешнего вида одного web-сайта, например, стационарной и «мобильной» версий.

Контроллер MVC-модели выполняет обработку и диспетчеризацию HTTP-запросов. Наибольшее распространение получили два следующих типовых решения:

1. при использовании паттерна «**контроллер страниц**» **(page controller)** задачи обработки HTTP-запросов и принятия решения о ее перенаправлении разделяются между собой. Функциями контроллера страниц являются:
    a. сбор сведений для выполнения запроса: анализ URL и извлечение данных, введенных пользователем в соответствующие формы;
    b. создание объектов модели и вызов их методов, необходимых для обработки данных. При этом все необходимые данные должны быть переданы модели из HTTP-запроса;
    c. определение представления, которое должно быть использовано для отображения результатов и передача ему необходимой информации, полученной от модели.

Контроллер страниц может быть реализован в виде сценария (CGI, сервлета и т.п.) или страницы сервера (ASP, PHP, JSP и т.п.). Использование страницы сервера обычно предполагает сочетание в одном файле **контроллера страниц** и **представления по шаблону**.

Как правило, **контроллер страниц** применяется для сайтов с достаточно простой логикой котроллера, поскольку предполагает наличие отдельного контроллера для каждой логической страницы Web-сайта. Если сайт имеет сложную структуру, а также, если предъявляются специальные требования к проверке безопасности, обеспечению интернационализации или открытию

специальных представлений для особых категорий пользователей, то использование котроллеров страниц приводит к дублированию кода. Дублирование кода можно избежать двумя способами: (i) посредством вынесения общей логики контроллеров в служебные классы, которые составят **супертип слоя**[3]; (ii) посредством использования типового решения **контроллер запросов**.

2. **контроллер запросов (front controller)** предполагает использование единственного объекта, предназначенного для обработки всех запросов, но включает в свой состав две части:
    a. web-обработчик – объект, который выполняет фактическое получение POST- и GET-запросов, поступивших на web-сервер. Он извлекает необходимую информацию из URL и входных данных запроса, после чего решает, какое действие необходимо инициировать, и делегирует его выполнение соответствующей команде.
    b. иерархия команд из числа которых web-обработчиком статически или динамически выбирается необходимая команда для обработки поступившего запроса. Статический выбор команды подразумевает проведение синтаксического анализа адреса URL и применение условной логики, а динамический – извлечение некоторого стандартного фрагмента адреса URL и динамическое создание экземпляра класса команды.

Web-обработчик интерпретирует полученный адрес URL и создает отдельный объект для дальнейшего обслуживания запроса. Таким образом удается централизовать деятельность по обработке всех HTTP-запросов в рамках единого объекта и избежать необходимости изменения конфигурации web-сервера в случае модификации функционала сайта.

Кроме того, контроллер запросов хорошо сочетается с типовым решением **перехватывающий фильтр**, который представляет собой **декоратор**[4], выступающий в роли оболочки web-обработчика котроллера запросов и позволяющий строить цепочки фильтров, предназначенные для обработки таких процессов как: аутентификация, регистрация на сайте, выбор кодировки и т.д. [74, 95].

Выбор типового решения для обеспечения взаимодействия «представление-контроллер» во многом зависит от используемой инструментальной среды. В частности, при использовании Visual Studio в большинстве случаев используется связка **«контроллер страниц – представление по шаблону**», а при использовании среды Struts(Java) - **«контроллер запросов – представление по шаблону**».

Контроллер страниц и контроллер запросов, получив название входных контроллеров, находят применение при проектировании web-сайтов, состоящих из нескольких десятков страниц. Если проектируемый сайт включает в состав 30 и более страниц логика выбора нужных экранов начинает дублироваться в нескольких контроллерах, что крайне нежелательно. В таком случае помимо входных контроллеров создается дополнительный сервисный слой – **контроллер приложения (application controller)**.

Контролер приложения выполняет две основные функции: выбор логики домена, которую нужно применить в конкретной ситуации, и выбор представления, которое следует отобразить в ответ на запрос. Для осуществления этих функций контроллер приложения должен поддерживать две коллекции ссылок на классы: одну для команд, выполняющихся в слое домена, и одну для представлений. В качестве команд домена могут выступать объекты команд, являющиеся частью слоя контроллера приложения, либо ссылки на сценарий транзакции или методы объектов домена.

Если в качестве представлений используются страницы сервера, то для вызова представления можно применить имя соответствующей страницы.

Подводя итог, остановимся на вопросе согласования рассмотренных выше типовых решений между собой (рис. 10).

Проектирование архитектуры слоев ИС начинается с выбора паттерна слоя предметной области, который определит типовое решение слоя источника данных. Как указывалось выше, **шлюз таблицы данных** гармонично сочетается с **модулем таблицы**: методы **шлюза таблицы данных** возвращают структуры данных в виде **множества записей**, которые затем обрабатываются **модулем таблицы**. Вместе с тем, **шлюз таблицы данных** может успешно использоваться и со **сценарием транзакции**. И в том и в другом случае выбор между **шлюзом записи данных** и **шлюзом таблицы данных** зависит лишь от предъявляемых требований к форме возвращаемого результата: в виде от-

[3] **Супертип слоя (layer supertype)** используется тогда, когда все объекты соответствующего слоя имеют некоторые общие свойства или поведение. В этом случае рекомендуется создать суперкласс для всех объектов слоя и вынести в него все общие методы для того, чтобы не дублировать их в объектах слоя.

[4] **Декоратор** – это паттерн, позволяющий динамически возлагать на объект новые функции.

дельной записи или в виде множества записей. Однако, при проектировании слоя предметной области с использованием сценария транзакций может оказаться, что в различных сценариях повторяется одна и та же бизнес-логика. В этом случае рекомендуется перенести бизнес-логику в шлюз записи данных, что превратит его в **активную запись**.

Паттерн «**активная запись**» рекомендуется использовать совместно с «простой» моделью предметной области, когда структура домена совпадает с ER-моделью базы данных. Если же модель предметной области является «сложной» единственным решением, позволяющим обеспечить полную независимость слоя бизнес логики от слоя предметной области, является «**преобразователь данных**».

Аналогичный анализ проводится и в отношении слоя представления. В том случае, если клиентская часть системы имеет множество экранов (страниц), то для организации алгоритма их вызова дополнительно вводится «**контроллер приложения**». В противном случае пользуются сочетаниями **«контроллер страниц – представление по шаблону**» или **«контроллер запросов – представление по шаблону**». Приобретение того или иного фирменного преобразователя (например, XSLT) позволит использовать «**представление с преобразованием**». К **двухэтапному преобразованию** прибегают при необходимости поддерживать нескольких вариантов внешнего вида web-сайта.

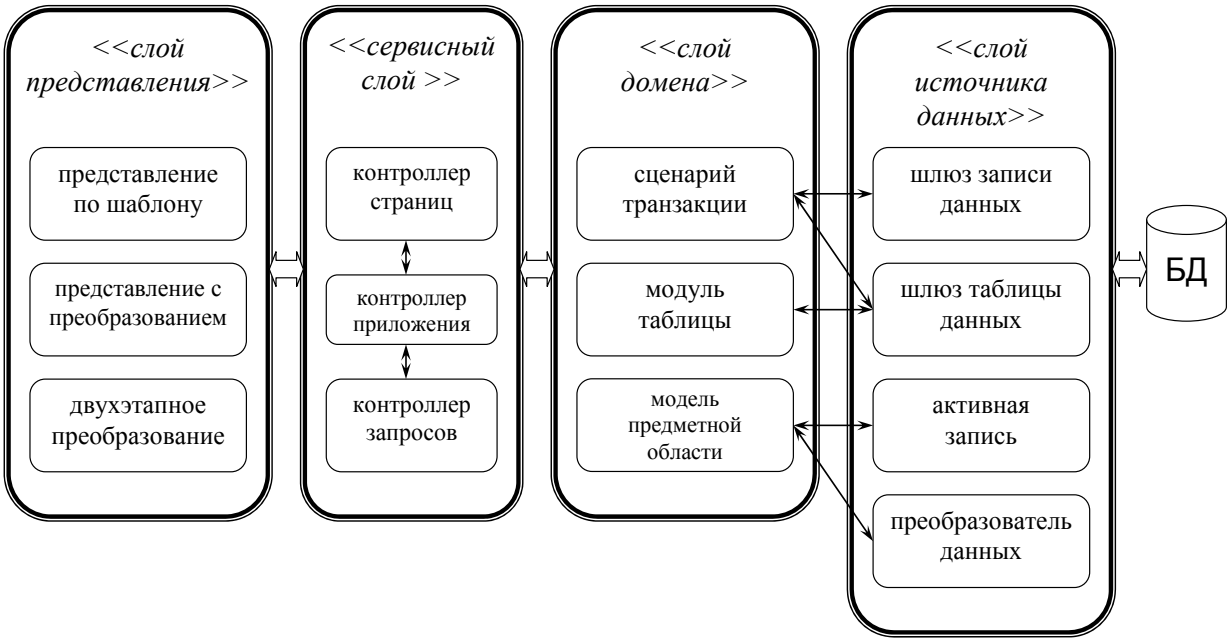

Рис. 10. Структура архитектурных слоев информационной системы

Таким образом, на страницах ПЗ в главе, посвященной разработке концепции АСОИУ (ИМС), необходимо:

1. **воспользовавшись литературными источниками [10, 14, 15, 34, 55] проанализировать наиболее подходящие для проектируемой ИС варианты архитектуры, чтобы выбрать один из них для дальнейшей проработки;**
2. **воспользовавшись паттернами GRASP и GoF [59, 62, 74, 80, 95] преобразовать классы анализа, полученные на этапе формулирования требований к ИС, в проектные классы. При этом в тексте ПЗ необходимо прокомментировать порядок использования того или иного паттерна;**

3. **воспользовавшись рекомендациями, изложенными в [98], аргументировано выбрать и описать (с использованием диаграмм UML) типовое решение для каждого из трех слоев проектируемой ИС: представления, предметной области и источника данных;**
4. **с использованием GoF-паттернов (абстрактная фабрика, мост, цепочка обязанностей, команда, фасад, посредник, наблюдатель) разместить все классы, относящиеся к тому или иному слою, в отдельные пакеты, обеспечив выполнение фундаментального принципа модульной декомпозиции «низкая внешняя связанность – высокое внутреннее сцепление»;**
5. **с использованием структурных GoF-паттернов (адаптер, заместитель, фасад) выполнить проектирование интерфейсов отдельных пакетов системы.**

Результаты, полученные при разработке концепции АСОИУ (ИМС), будут детализированы на стадии эскизный проект и опубликованы в виде проектной документации программного обеспечения (SDD – Software Design Document) в приложении к ПЗ.

4.9. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Техническое задание (ТЗ) на АСОИУ является основным документом, определяющим требования и порядок создания (развития или модернизации) автоматизированной системы, в соответствии с которым проводится разработка АСОИУ и ее приемка при вводе в эксплуатацию. ТЗ включает в себя системное описание расширенных требований к разрабатываемому изделию и составляется на основе исходного задания по курсовому проекту с учетом всей проведенной выше работы в отношении формулирования и спецификации требований.

ТЗ на АСОИУ разрабатывают на систему в целом, предназначенную для работы самостоятельно или в составе другой системы. Дополнительно могут быть разработаны ТЗ на части АСОИУ (на подсистемы АСОИУ, комплексы задач АСОИУ и т. п.; на комплектующие средства; на программные средства; на информационные изделия и т.д.) [71].

Включаемые в ТЗ на АСОИУ требования должны соответствовать современному уровню развития науки и техники и не уступать аналогичным требованиям, предъявляемым к лучшим современным отечественным и зарубежным аналогам. Задаваемые в ТЗ на АСОИУ требования не должны ограничивать разработчика системы в поиске и реализации наиболее эффективных технических, технико-экономических и других решений.

ТЗ на АСОИУ разрабатывают на основании исходных данных, а также данных, содержащихся в итоговой документации стадии «Формирование требований к АСОИУ».

В ТЗ на АСОИУ включают только те требования, которые дополняют требования к системам данного вида (АСУ, САПР, АСНИ и т. д.), содержащиеся в действующих НТД, и определяются спецификой конкретного объекта, для которого создается система.

Поскольку в подразделе 2.6.1. ТЗ на создание АСОИУ (ГОСТ 34.602-89) необходимо перечислить требования к системе в целом, а в подразделе 2.6.3 требования к различным видам обеспечения, то в главе «Техническое задание» пояснительной записки необходимо с учетом специфики проектируемой АСОИУ, а также данных, указанных в задании на курсовой проект, выполнить анализ требований, имеющих наиболее существенное значение.

В данном разделе ПЗ могут быть проанализированы требования к структуре и функционированию системы, к численности и квалификации персонала системы и режиму его работы, показатели назначения, требования к надежности, безопасности, эргономике и технической эстетике, к эксплуатации, защите информации от несанкционированного доступа, требования по сохранности информации при авариях, требования к защите от влияния внешних воздействий, к патентной чистоте, требования по стандартизации и унификации; а также требования к видам обеспечения: математическому, информационному, лингвистическому, программному, техническому, метрологическому, организационному, методическому и другим [66, 68].

В частности, при описании требований к информационному обеспечению в большинстве случаев анализируют системы управления базами данных, структуру процессов сбора, обработки, передачи данных в системе и представлению данных, защите данных от разрушений при авариях и сбоях в электропитании системы. При описании требований к лингвистическому обеспечению рассматривают требования к применению в системе языков программирования высокого уровня, языков взаимодействия пользователей и технических средств системы, а также требования к кодированию и декодированию данных, к языкам ввода-вывода данных, языкам манипулирования данными, средствам описания предметной области (объекта автоматизации). При описании требований к программному обеспечению системы приводят перечень покупных программных средств, а также требования к независимости программных средств от используемых СВТ и операционной среды, к качеству программных средств, а также к способам его обеспечения и контроля.

Техническое задание на создание АСОИУ, оформленное в соответствии с ГОСТ 34.602-89 «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы», помещается в приложение ПЗ. Учебный пример составления ТЗ см. в [60, 76].

4.10. ЭСКИЗНЫЙ ПРОЕКТ

В главе «Эскизный проект» описывается детальное проектирование системы. Целью детального проектирования является создание проектной документации, по которой может быть создана система без какой бы то ни было дополнительной информации (возможно силами другой организации). Как указывалось в разделе «План управления программным проектом» (п. 4.7.1.5) начинать процесс детального проектирования необходимо с аспектов проекта, представляющих наибольший риск.

Выполнение детального проектирования системы предполагает проработку следующих шагов:

1. Введение глобальных пакетов:
   1.1. базисные (foundation) классы (списки, очереди и т.д.);
   1.2. обработчики ошибок (error handling classes);
   1.3. математические библиотеки;
   1.4. утилиты;
   1.5. библиотеки других поставщиков.
2. Детализация архитектурных слоев.
   Проанализированные в главе «Разработка концепции АСОИУ» типовые решения архитектурных слоев ИС, как правило, нуждаются в детализации, поскольку вошедшие в состав системы паттерны изначально имеют существенно упрощенную структуру, в которой множество условий и деталей были пропущены для улучшения понимания назначения паттерна.
3. Уточнение проектных классов (design classes):
   3.1. Уточнение атрибутов класса:
      3.1.1. кроме имени атрибута, задается его тип и значение по умолчанию;
      3.1.2. учитываются соглашения по именованию атрибутов, принятые в проекте и языке реализации;
      3.1.3. задается видимость атрибутов: public, private или protected;
      3.1.4. при необходимости определяются производные (вычисляемые) атрибуты.
   3.2. Уточнение операций класса:
      3.2.1. создается краткое описание операции, включая пояснение смысла всех ее параметров;
      3.2.2. определяется видимость операции: public, private или protected;
      3.2.3. определяется область действия операции: экземпляр (операция объекта) или классификатор (операция класса);
      3.2.4. задаются предусловия функции (при каких условиях выполняется). Предусловия описывают соотношения между переменными и константами, существование которых предполагается до момента выполнения функции;
      3.2.5. задаются постусловия функции (что происходит в результате ее выполнения);
      3.2.6. составляется описание алгоритма выполнения операции (с использованием диаграмм деятельности, блок-схем, псевдокода, а также диаграмм взаимодействия различных объектов при выполнении операции).

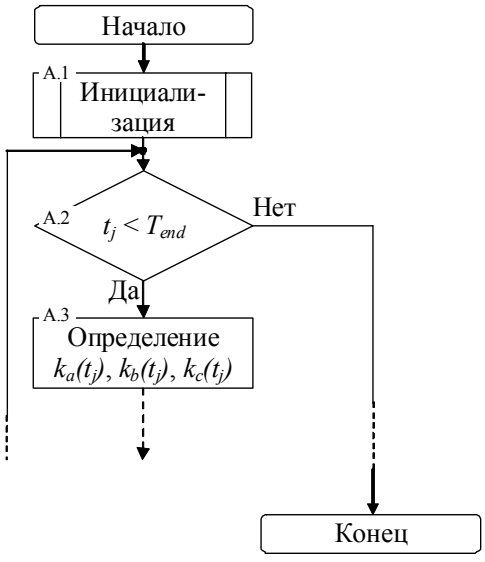

Рис. 11. Пример оформления блок-схемы алгоритма

Фрагмент некоторого алгоритма, представленного в виде блок-схемы, показан на рис. 11. Каждый блок алгоритма снабжен идентификатором, который используется для ссылки при описании выполняющихся в нем операторов. Блок-схемы алгоритмов приводят в приложении к ПЗ и оформляют в соответствии с ГОСТ 19.003-80 «Схемы алгоритмов и программ. Обозначения условные графические» [63].

3.3. Определение инвариантов класса.

Инварианты класса представляют собой эффективные требования к объектам класса, выраженные в терминах атрибутов класса. Инварианты класса принимают форму ограничений на значения. Как и С-требования, они часто называются бизнес-правилами. Например, требование «Все участники веб-аукционов должны предоставлять номер своей кредитной карты»

может быть переведено в специальный инвариант класса Участник следующим образом:
зарегистрирован == true AND 400000001 <= номерКредитнойКарты <= 699999999 OR зарегистрирован == false AND номерКредитнойКарты == 0.

4. Проработка интерфейсов между слоями проектируемой системы или параллельными процессами.

Поскольку вопросы проектирования интерфейсов играют ключевую роль в обеспечении производительности при распределенной обработке данных в ИС, несколько типовых решений организации интерфейсов будут рассмотрены в следующем параграфе.

4.10.1. Организация программных интерфейсов

Организация одновременного функционирования нескольких процессов в рамках одной информационной системы становится все более и более актуальной задачей. Если несколько процессов одновременно выполняются на одном узле, то рассматриваются вопросы организации параллельных вычислений. Если же процессы запущены на разных узлах вычислительной сети и для достижения поставленной цели необходимо организовать обмен данными между ними, то рассматривается задача распределенных вычислений (или распределенной обработки данных). Наиболее очевидный пример распределенной обработки – организация взаимодействия между клиентом и сервером корпоративного приложения.

Наиболее существенная проблема организации распределенных приложений связана с минимизацией временных издержек, вызванных передачей данных по каналам связи между удаленными узлами сети. При этом, поскольку каждый удаленный вызов приводит к увеличению временных затрат, то при проектировании распределенных приложений стремятся заменить множество «мелких» (в смысле объема возвращаемых данных) вызовов несколькими «крупными» вызовами, возвращающими возможно избыточную в данный момент, но потенциально необходимую информацию для выполнения последующих этапов вычислений. Реализовать такую идею можно путем введения дополнительных объектов с низкой степенью детализации интерфейса, назначение которых состоит в организации доступа к сети объектов с высокой степенью детализации интерфейса (например, модели предметной области). Для этого применяется типовое решение, получившее название **интерфейса уделанного доступа (remote facade)**.

На рис. 12 приведен простой пример организации интерфейса удаленного доступа: многочисленные set- и get-методы объекта *Book* интерфейсом удаленного доступа *BookFacade* заменяются на одну пару set-get методов для всего объекта в целом. В этом случае, если, например, *Удаленный клиент* обращается к методу *setBookData*, интерфейс удаленного доступа считывает параметры, переданные в удаленном запросе, и последовательно вызывает отдельные set-методы объекта *Book*.

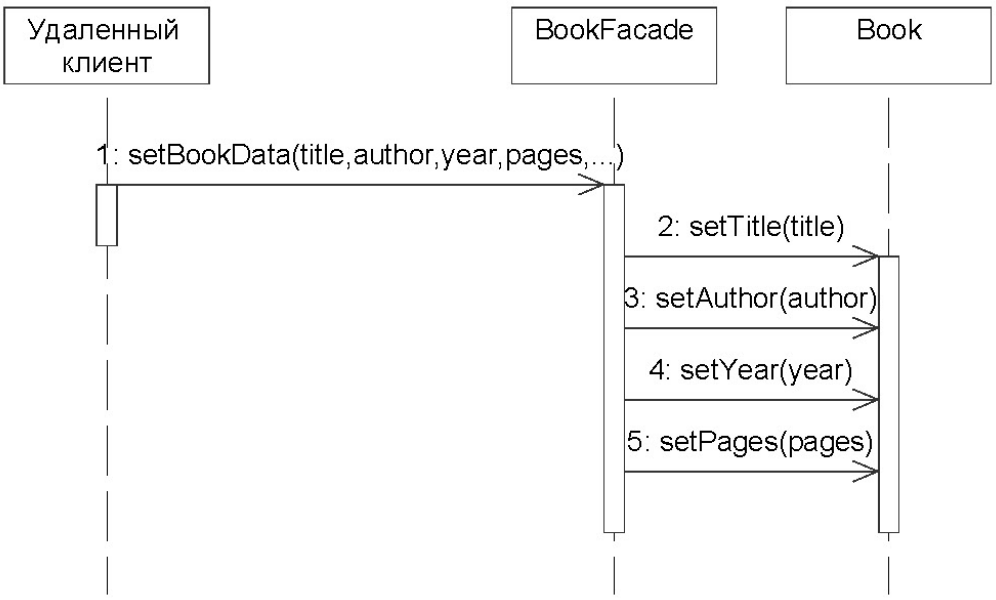

Рис. 12. Пример организации интерфейса удаленного доступа

Помимо предоставления интерфейса с низкой степенью детализации, **интерфейс удаленного доступа** может выполнять и другие функции, например, обеспечение безопасности или управление транзакциями[5].

Если взаимодействие с клиентами требует сохранения состояния объектов между сеансами необходимо использовать одно из следующих типовых решений: **сохранение состояния сеанса на стороне клиента**, **сохранение состояния сеанса в БД**, **сохранение состояния сеанса на стороне сервера**.

В более сложных случаях, чем в рассмотренном выше примере (см. рис. 12), один интерфейс удаленного доступа может выполнять роль шлюза для нескольких объектов с высокой степенью детализации. Например, интерфейс удаленного доступа для объекта *Заказ* может применяться для извлечения и обновления сведений о заказе, всех его пунктах, а также некоторых сведений о покупателе, разместившем этот заказ. При этом клиенту может потребоваться, например, получить с сервера связный граф объектов, относящихся к объектам предметной области. Для реализации такой передачи применяется **объект переноса данных (data transfer object)**, возвращаемый клиенту за один вызов, и содержащий не только те данные, которые запросил клиент, но и те, которые ему могут понадобиться на следующем этапе взаимодействия с системой.

Поля объектов переноса данных, как правило, содержат значения стандартных типов (строки, даты и т.д.), а также другие объекты переноса данных. Любые связи между такими объектами должны укладываться в рамки простого графа, типа иерархии, в противоположность сложным структурам модели предметной области. Поэтому для переноса данных обычно используют упрощенную форму объектов домена.

Наиболее распространенной формой реализации объекта переноса данных является множество записей – набор табличных записей, который возвращается в результате выполнения SQL-запроса, сгенерированного доменом.

С целью «разворачивания» **объектов переноса данных** на обоих концах соединения используется специальный объект, получивший название «**сборщик**», который создает **объект переноса данных** на основе данных домена и наоборот – обновляет модель домена данными из **объекта переноса данных**. При создании объекта переноса данных **сборщиком** используется тот или иной алгоритм сериализации, позволяющий получать данные в двоичном, текстовом (причем, в большинстве случаев, в виде XML-документа) или в зашифрованном формате.

Помимо проработки вопросов, связанных с обеспечением интерфейса удаленного доступа, необходимо предусмотреть интерфейс для любого источника или получателя данных в системе. Такие интерфейсы чаще всего называются **шлюзом (gateway)** и помимо доступа, например, к реляционной базе данных (такой частный случай рассмотрен в разделе 4.8.2) могут предоставлять доступ к XML-документам, транзакциям CICS[6], W3C[7], JDOM[8] и т.д.

В большинстве случаев **шлюз** представляет собой класс в который помещается весь специализированный код API того или иного источника или потребителя данных. Интерфейс **шлюза** соответствует интерфейсу обычного объекта доступа к данным и может содержать набор традиционных методов. При этом для получения доступа к удаленному источнику, объекты приложения будут обращаться к шлюзу, который в свою очередь будет преобразовывать вызовы простых методов в вызовы специализированного API. Очевидно, что для перехода к другому источнику данных в этом случае достаточно будет изменить только класс шлюза, а остальная часть системы абсолютно не пострадает.

Следует подчеркнуть, что согласно идеологии применения целого ряда GoF-паттернов (**Indirection, Protect Variation, Adapter**) существует объективная необходимость применения служебных классов типа «**шлюз**» для организации взаимодействия между различными подсистемами с целью инкапсуляции их внутренней структуры, посредством обеспечения доступа к высокоуровневым методам, вынесенным в интерфейсную часть. Такой подход позволяет обеспечить фундаментальный

---

[5] В большинстве случаев под транзакциями подразумеваются последовательности операций, описывающих процессы взаимодействия с БД, однако множество других объектов управляются с помощью механизма поддержки транзакций: очереди сообщений, задания на печать, банковские платежи и т.д. Поскольку транзакции являются основным инструментом управления параллельными процессами в корпоративных приложениях широко известны несколько алгоритмов диспетчеризации транзакций: длинные транзакции, транзакции запроса, отсроченные транзакции и т.д.

[6] CICS (Command Execution Diagnostic Facility) – среда обработки транзакций фирмы IBM

[7] W3C - World Wide Web Consortium

[8] JDOM - Java-реализация популярного интерфейса доступа к документам (Document Object Model - DOM)

принцип модульной декомпозиции «низкая внешняя связанность – высокое внутреннее сцепление», а также повысить эффективность тестирования отдельных компонентов за счет сокращения количества тестовых наборов.

Пример организации двухуровневой архитектуры корпоративного приложения типа «клиент-сервер» показан на рис. 13. Помимо узкоспециализированных шлюзов, предназначенных для взаимодействия с БД, *слой источника данных* содержит **шлюз**, обеспечивающий доступ к внешним источникам или потребителям данных. Кроме того, в *слое домена* выделены подсистемы, выполняющие независимые друг от друга задачи, и взаимодействующие друг с другом посредством межкомпонентного интерфейса. При поступлении удаленного вызова от клиента **интерфейс удаленного доступа** с помощью **сборщика** создает **объект переноса данных**, который используется для обмена информацией между клиентом и сервером.

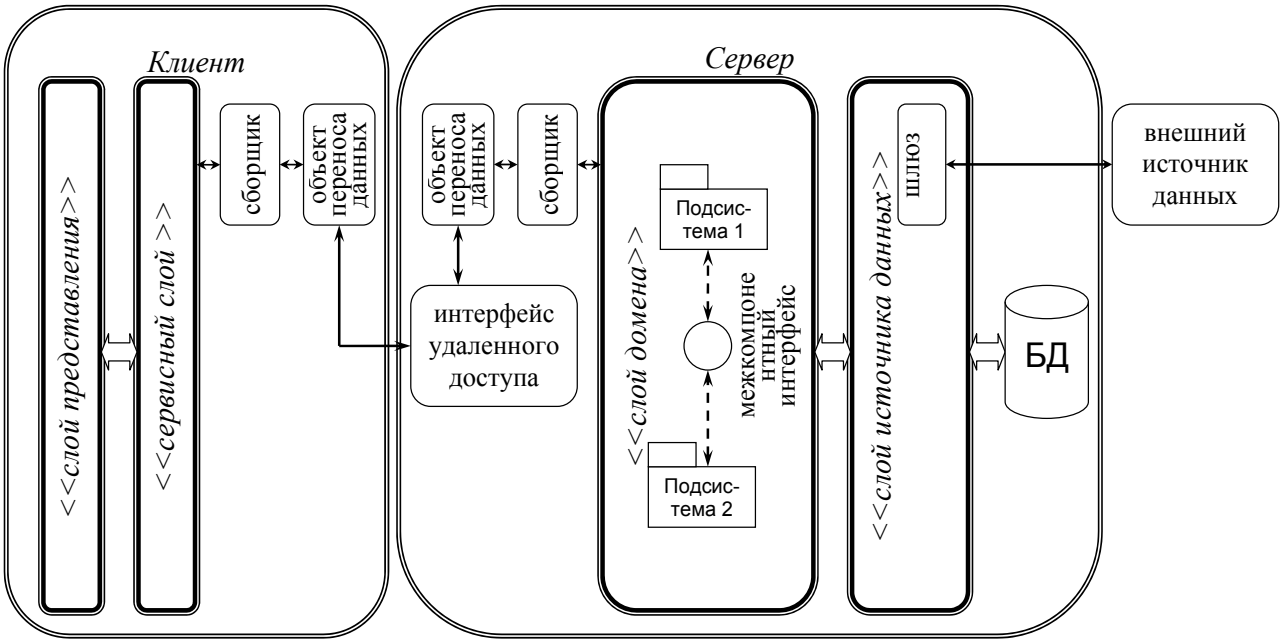

Рис. 13. Пример организации двухуровневой архитектуры типа «клиент-сервер»

Типовые решения по организации межслойных и межмодульных интерфейсов широко обсуждаются в литературе. В частности, в [62, 80, 95, 98] описаны основные концепции организации интерфейсов и приведены примеры их реализации.

4.10.2. Проектирование структуры базы данных

В заключение главы «Эскизный проект» разрабатывают структуру базы данных (БД).

Проектирование БД зависит от типа используемой для хранения данных СУБД – объектной или реляционной. Для объектных БД проектирования не требуется, поскольку классы-сущности непосредственно отображаются в БД. Для реляционных БД классы-сущности объектной модели должны быть отображены в таблицы реляционной БД. Совокупность таблиц и связей между ними может быть представлена в виде диаграммы классов, которая по существу является ER-диаграммой. Набор правил, применяемых при отображении классов в таблицы БД, фактически совпадает с правилами преобразования сущностей и связей.

Для описания схемы БД применяется следующий набор элементов языка UML:

- таблица представляется в виде класса со стереотипом «Table»;
- представление изображается в виде класса со стереотипом «View»;
- столбец таблицы представляется в виде атрибута класса с соответствующим типом данных;
- обычная ассоциация и агрегация представляются в виде ассоциации со стереотипом «Non-Identifying» (в терминологии IDEF1X – неидентифицирующая связь);
- композиция представляется в виде ассоциации со стереотипом «Identifying» (в терминологии IDEF1X – идентифицирующая связь);
- схема БД представляется в виде пакета со стереотипом «Schema», содержащего классы-таблицы;
- контейнер хранимых процедур представляется в виде класса со стереотипом «SP Container»;

- ограничения целостности, индексы и триггеры представляются в виде операций классов-таблиц со стереотипами «PK» (Primary Key), «FK» (Foreign Key), «Unique», «Check», «Index» и «Trigger»;
- физическая база данных представляется в виде компонента со стереотипом «Database».

Кроме создания структуры при проектировании БД, как правило, перечисляются все запросы, которые необходимо предусмотреть в системе для ее полноценного функционирования, а также приводят синтаксис и описание типовых SQL-запросов, хранимых процедур, представлений и триггеров, позволяющих, с одной стороны, осуществлять мониторинг за состоянием БД, с другой стороны, решить задачу построения сложно-структурированных запросов к БД, тем самым повысив надежность и расширяемость системы, а также упростив программирование клиентской части приложения.

Стандарт IEEE 1016-1987 для проектной документации программного обеспечения (SDD – Software Design Document) содержит руководство по составлению и ведению проектной документации системы. Основные результаты проектирования архитектуры, а также результаты, полученные в эскизном проекте, помещаются в соответствующий раздел SDD.

Подводя итог, еще раз перечислим требования к содержанию главы «Эскизный проект» пояснительной записки:

1. **детализировать архитектурные слои и описать основные и вспомогательные пакеты, вошедшие в их состав;**
2. **детализировать и описать проектные классы (указать инварианты, операции, атрибуты, область их видимости, пред- и пост- условия функций). Для функции наиболее трудоемкой с точки зрения реализации составляется блок-схема алгоритма выполнения операций;**
3. **с использованием типовых решений «интерфейс удаленного доступа», «объект переноса данных», «сборщик» и т.д. прорабатываются межслойные интерфейсы проектируемой системы;**
4. **разрабатывается структура БД, прорабатываются типовые запросы, триггеры, представления и хранимые процедуры, которые необходимо предусмотреть в системе для ее полноценного функционирования.**

Проектная документация системы, оформленная в соответствии с IEEE 1016-1987, помещается в приложение ПЗ. Пример составления проектной документации см. в [55]. Оглавление проектной документации программного обеспечения приведено в приложении Л и включает в качестве примера типовое описание некоторых разделов, которое необходимо адаптировать в соответствии с содержанием выполняемого проекта. Примечание к тому или иному разделу приведено в квадратных скобках курсивным шрифтом.

4.10.3. Интеграция приложений

В проектах, состоящих из нескольких независимых частей, в особенности, в комплексных проектах, выполнение которых осуществляется несколькими студентами, необходимо особое внимание уделить вопросам интеграции. Актуальность поиска и проработки интеграционных решений связана со следующими обстоятельствами:

1. **Ненадежность сети передачи данных.** Распределенные системы могут быть связаны друг с другом посредством телефонных линий, сегментов локальных сетей, маршрутизаторов, коммутаторов, спутниковых каналов связи и т.д. Доставка информации на каждом из этих этапов связана с задержкой и риском потери.
2. **Низкая скорость передачи данных.** Время передачи данных через каналы связи на порядок больше вызова локальной процедуры. В этой связи, создание распределенного приложения требует применение иных принципов проектирования, чем создание приложения, выполняющегося локально на одном компьютере.
3. **Различия между приложениями.** Интеграционное решение должно быть ингерентным к различиям составляющих его частей (язык программирования, платформа, формат данных и т.д.).
4. **Подверженность изменениям.** Чем больше в состав приложения входит различных составляющих, тем в большей степени оно будет подвержено неминуемым изменениям. В этой связи интеграционное решение должно иметь возможность адаптации к изменению объединяемых им приложений.

Для устранения (или снижения влияния) указанных выше проблем используются четыре типовых решения, предназначенных для обмена информацией между приложениями, входящими в состав интеграционной системы:

1. Обмен информацией посредством передачи файлов (**file transfer**);
2. Обмен информацией посредством использования общей базы данных (**shared database**);
3. Приложения предоставляют доступ к части своей функциональности посредством удаленного вызов процедуры (**remote procedure invocation** или **remote facade** (см. п. 4.10.1));
4. Обмен информацией между приложениями осуществляется посредством обмена сообщениями (**messaging**).

По мнению специалистов [101] наиболее перспективным направлением интеграции приложений является технология высокоскоростного асинхронного взаимодействия посредством обмена сообщениями, которая позволяет обеспечить более скоростной обмен информацией, чем передача файлов, обладает лучшей инкапсуляцией по сравнению с общей базой данных и является более надежной, чем удаленный вызов процедуры.

На страницах ПЗ необходимо с использованием литературных источников [24, 101] проанализировать типовые решения по интеграции корпоративных приложений и описать детали проектных решений, принятых в работе.

4.11. ЗАКЛЮЧЕНИЕ

Корректное заключение позволяет судить о правильности постановки задачи, ходе работы и о полученном результате. Заключение должно быть информативным – оно должно в сжатом виде дать читателю информацию, причем, настолько полную, чтобы можно было не изучать текст пояснительной записки подробно.

Выводы заключения должны приводиться в последовательности, соответствующей их значимости: первым должен быть указан наиболее глобальный вывод, отражающий главный результат работы, а последующие должны его развивать и уточнять.

Выводы должны быть информативными, т.е. должны нести информацию о сути, взаимосвязях, физической трактовке взаимодействия изученных факторов.

Выводы заключения должны иметь прямую связь с целью работы и ее основными задачами.

При подготовке заключения наиболее распространенной ошибкой является подмена выводов информацией о проделанной работе. В этой связи при подготовке заключения нельзя допускать в тексте такие фразы как: «проведен анализ…», «сформулированы требования» и т.д., ограниченные только названием и не наполненные раскрытием сути, смысла соответствующих понятий.

Примеры корректных и неполных выводов, а также рекомендации по составлению заключения приведены в [102].

4.12. СПИСОК ЛИТЕРАТУРЫ

Список литературы оформляется согласно требованиям ГОСТ 7.0.5-2008 «Библиографическая ссылка. Общие требования и правила составления» [72] и включает в себя перечень литературных источников в алфавитном порядке, на которые в тексте ПЗ имеются ссылки.

4.13. ПРИЛОЖЕНИЯ

Минимальный перечень документов, которые необходимо вынести в приложение к пояснительной записке включает:

1. план управления конфигурациями (IEEE 828-1990);
2. план контроля качества программного обеспечения (IEEE 730-1989);
3. план управления программным проектом (IEEE 1058.1-1987);
4. спецификация требований к программному обеспечению (IEEE 830-1993);
5. техническое задание (ГОСТ 34.602-89);
6. проектная документация программного обеспечения (IEEE 1016-1987);
7. отчет об инспектировании.

Приложения располагаются в порядке ссылок в тексте документа [65].

Рекомендуется в приложение к пояснительной записке помещать рисунки, ширина которых не превышает 210 мм, а высота больше 145 мм.

5. ОФОРМЛЕНИЕ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ ПО КУРСОВОМУ ПРОЕКТУ

Структура каждого раздела ПЗ непременно должна включать в свой состав: (i) небольшую вводную часть (2-3 абзаца), в которой необходимо раскрыть цель раздела и сделать соответствующие определения; (ii) основную часть, включающую в себя рассмотрение вопросов, непосредственно связанных с данным разделом; (iii) в завершении необходимо, подводя итоги, сделать выводы или обобщить результаты, полученные в разделе.

Текст пояснительной записки оформляется в строгом соответствии с ГОСТ 2.105-95 «Общие требования к текстовым документам» [65], помимо которых необходимо соблюдать следующие требования:

1. весь тест пояснительной записки выполняется исключительно одним шрифтом (Times New Roman, размер 12 пунктов, одинарный межстрочный интервал, выравнивание абзацев по ширине, абзацные отступы – 1,25 или 1,27 см, автоматическая расстановка переносов). Допускается использование прописных букв в заголовках;
2. поля страницы: левое – 25 мм, правое – 10 мм, верхнее и нижнее – по 20 мм, нумерация страниц в правом верхнем углу;
3. высота букв надписей в таблицах и на диаграммах не должна быть меньше высоты букв основного текста;
4. каждый элемент, изображенный на диаграмме (рисунке), должен быть упомянут в тексте при описании, которое, как правило, предшествует рисунку.

6. ПОРЯДОК ВЫПОЛНЕНИЯ И ЗАЩИТЫ КУРСОВОГО ПРОЕКТА

В течение первых 10 дней от начала семестра необходимо получить и согласовать задание по КП с руководителем. Если по истечении 10 дней от начала семестра студентом не была предложена тема КП, типовая тема назначается руководителем. Формулировка назначенной темы КП, дата начала проектирования, дата сдачи работы на рецензирование, а также исходные данные уточняются в задании на КП в момент его подписания. Кроме того, в течение первых 10 дней каждый студент согласовывает с руководителем кандидатуру инспектора, в роли которого выступает студент этой же группы. По окончании 10 дней при отсутствии предложений со стороны студента-исполнителя, студент-инспектор назначается руководителем КП по списку группы.

На рис. 14 приведена UML диаграмма деятельности, демонстрирующая порядок подготовки и защиты КП. Из диаграммы деятельности видно, что сразу после согласования задания студент приступает к подготовке ПЗ. По каждой главе выполняется сбор и анализ литературных источников, формулируются и описываются результаты. После выполнения каждого раздела курсового проекта студент-исполнитель передает логически-завершенный, оформленный, согласно требованиям настоящих методических указаний, комплект документов студенту-инспектору и сам выполняет инспектирование подготовленной части работы своего коллеги, для которого он назначен «опекуном». По окончании работы над проектом студент-инспектор помещает отчет об инспектировании в приложение к своей ПЗ (образец оформления титульного листа отчета см. в приложении М). Каждый законченный раздел пояснительной записки или результаты очередной итерации представляются руководителю курсового проекта для обсуждения на консультациях в соответствии с графиком, установленном кафедрой.

Законченная и оформленная в соответствии с требованиями настоящих МУ пояснительная записка по курсовому проекту распечатывается, скрепляется скоросшивателем, подписывается студентом и представляется руководителю для рецензирования не позднее, чем за 6 дней до защиты. Нарушение этого условия означает перенос даты защиты на период пересдач академических задолженностей.

Срок выполнения рецензирования не должен превышать трех дней. В результате рецензирования представляется подробный анализ недостатков и ошибок, уровень соответствия требованиям, формулируется список замечаний, которые студенту необходимо исправить.

На исправление замечаний отводится два дня. За один день до защиты студент передает чистовую версию ПЗ руководителю для проведения итогового рецензирования, по результатам которого принимается решение о назначении оценки за ПЗ. Если оценка за ПЗ оказывается неудовлетворительной (количество ошибок и погрешностей позволяют отнести ее к низкому уровню соответствия требованиям (см. табл. 8)), а также, если студент не успевает в установленный срок исправить указанные руководителем замечания, то дата защиты переносится на период пересдач академических задолженностей.

В случае если оценка за ПЗ оказывается положительной, то студент допускается к защите, о чем руководитель делает надпись на титульном листе ПЗ.

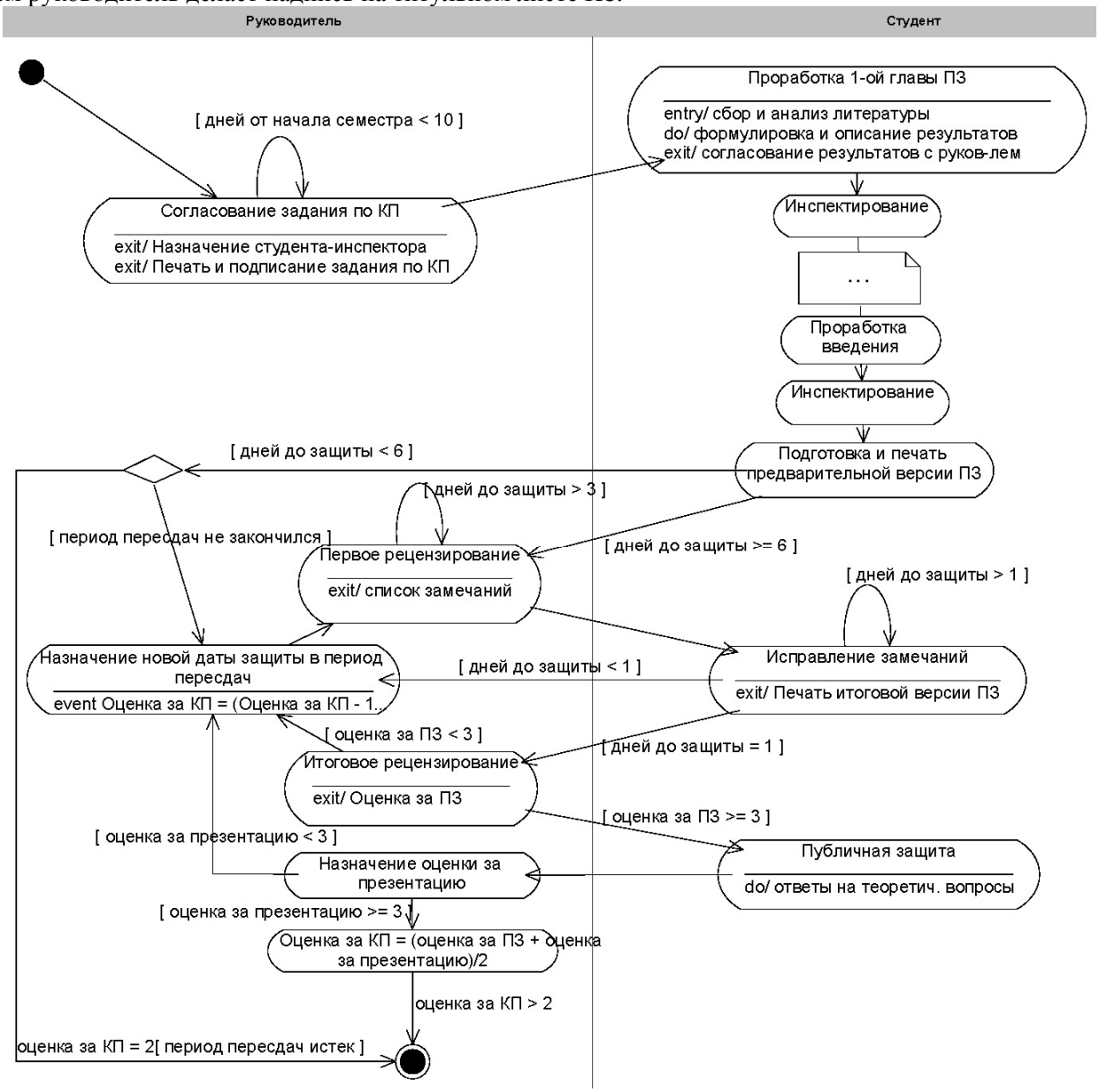

Рис. 14. UML диаграмма подготовки и защиты курсового проекта

Таблица 8

Сопоставление уровня соответствия требованиям со шкалой оценивания

Шкала соответствия	Уровень соответствия	Баллы	Количество ошибок: погрешности / несущественные ошибки / существенные ошибки
Соответствие требованиям	Высокий	5	4/0/0
	Средний	4	0/1/0 | 5/0/0
			0/2/0 | 5/1/0 | 10/0/0
			0/3/0 | 5/2/0 | 10/1/0 | 15/0/0 | 0/0/1
	Минимально необходимый	3	0/4/0 | 5/3/0 | 10/2/0 | 15/1/0 | 20/0/0 | 5/0/1 | 0/1/1
			0/5/0 | 5/4/0 | 10/3/0 | 15/2/0 | 20/1/0 | 25/0/0 | 10/0/1 | 5/1/1 | 0/2/1
			0/6/0 | 5/5/0 | 10/4/0 | 15/3/0 | 20/2/0 | 25/1/0 | 30/0/0 15/0/1 | 10/1/1 | 5/2/1 | 0/3/1 | 0/0/2
Несоответствие требованиям	Низкий	2	> 0/0/2 | …

Защита студентом КП проходит публично в виде презентации в присутствии своих коллег. Итоговая оценка за презентацию складывается из оценки за выступление (см. ниже требования к презентации) и оценки за ответы на три вопроса по теме выполненной работы. Если студент неудовлетворительно подготовил презентацию или не ответил ни на один вопрос, то защита переносится на период пересдач академических задолженностей.

Любая из указанных выше причин, повлекших перенос даты защиты на период пересдач академических задолженностей, означают снижение на один балл итоговой оценки за КП.

Итоговая оценка за курсовой проект представляет собой среднее арифметическое (с округлением до большего целого) оценок, полученных за пояснительную записку и презентацию проекта.

Примечание.

1. Погрешностями при определении степени соответствия курсового проекта требованиям считаются:
   1.1. неточные, двусмысленные, неполные формулировки и выражения в пояснительной записке;
   1.2. нерациональные, но правильные приемы обработки информации;
   1.3. незначительные погрешности при проектировании диаграмм;
   1.4. ошибки синтаксиса языка моделирования UML в именах объектов их свойств и методов.
2. К несущественным ошибкам относятся:
   2.1. неточности определения типов полей базы данных и переменных;
   2.2. неточности определения параметров функций и типов возвращаемых значений;
   2.3. неточности проектирования алгоритмов;
   2.4. нерациональный способ решения задачи или план ответа (нарушение логики изложения материала, подмена основных понятий второстепенными);
   2.5. отсутствие ссылок на использованные источники;
   2.6. несоблюдение требований ГОСТ в части оформления пояснительной записки и приложений.
3. К существенным ошибкам относятся:
   3.1. несоответствие результатов, изложенных в ПЗ, требованиям задания по КП;
   3.2. несоблюдение требований МУ в части содержания ПЗ;
   3.3. игнорирований замечаний руководителя, указанных им по результатам рецензирования;
   3.4. отсутствие прослеживания требований к проектируемой системе, перечисленных в задании на КП, при отражении результатов проектирования в основных разделах ПЗ: «формирование требований», «разработка концепции», «техническое задание», «эскизный проект»;
   3.5. подмена понятий в изложении основных категорий языка моделирования UML: классы, свойства класса, методы класса, объекты и т.д.;
   3.6. незнание фундаментальных понятий и категорий АСОИУ и теории проектирования АСОИУ: жизненный цикл, стадии создания, технология проектирования, методология проектирования и т.д.;
   3.7. неумение применять теоретические знания для решения задачи проектирования АСОИУ.

Требования к презентации

Защита проекта производится публично в присутствии студентов данной группы. Защита состоит в презентации студентом содержания выполненного проекта и в ответах на вопросы. Продолжительность выступления строго ограничена и не должна превышать 7 мин. Количество слайдов презентации не должно превышать 20-22. Примерное содержание презентации по защите курсового проекта приведено в таблице 9.

Таблица 9

Примерное содержание презентации по защите курсового проекта

№ слайда	Содержание слайда
1	Титульный лист (название организации, название доклада, ФИО выступающего, дата защиты …)
2,3	Постановка проблемы, обуславливающей актуальность темы проекта
4,5	1. Объект курсового проекта (автоматизация того или иного технологического процесса; направление информационной технологии апробации которой посвящена работа); 2. Предмет курсового проекта (приложение информационных технологий к решению

№ слайда	Содержание слайда
	конкретной прикладной задачи, относящейся к объекту проектирования, и обуславливающей актуальность представляемой работы) 3. Цель курсового проекта (повысить эффективность … путем …; упростить …, посредством …; улучшить надежность, масштабируемость, качество, расширяемость, безопасность … путем … и т.д.) 4. Перечень задач, решение которых позволит достичь поставленной цели.
6,7	Результаты и выводы по первой главе
8,9	Результаты и выводы по второй главе
…	…
19	Заключение
20	Дальнейшие (целесообразные) направления развития проекта

Ниже перечислены общие требования к презентации:

1. Слайды презентации должны быть выполнены в едином стиле. Все пространство слайда должно быть по возможности заполнено информацией. Надписи, в особенности подписи на рисунках, должны быть выполнены шрифтом с как можно более крупным кеглем.
2. Все то, что автор посчитал нужным привести на слайде, в тексте выступления должно быть разъяснено (сообщены дополнительные сведения, сделаны комментарии). Однако дословно дублировать текст, приведенный на слайде, в выступлении не имеет смысла.
3. На слайдах о результатах и выводах по той или иной главе необходимо приводить только ту информацию, которая была получена непосредственно автором в процессе выполнения проекта.

СПИСОК ЛИТЕРАТУРЫ

1. «Advanced Software Engineering: Expanding the Frontiers of Software Technology», IFIP 19th World Computer Congress. (2006 ; Santiago, Chile). First International Workshop on Advanced Software Engineering, 26-28 April 2006 / Edited by S.F. Ochoa. – New York : Springer, 2006. – 178 p.
2. «Advances in Databases and Information Systems», ADBIS 2008. (2008 ; Pori, Finland). 12th East European Conference, 5-9 September 2008 / Edited by Hannu Jaakkola. – Berlin : Springer, 2008. – 331 p.
3. «Agile Processes in Software Engineering and Extreme Programming», 8th International Conference, XP 2007 (2007 ; Como, Italy). 18–22 June 2007 / Edited by Giancarlo Succi. – Germany : Springer-Verlag Berlin Heidelberg, 2007. – 289 p.
4. «Database and XML Technologies», XSym 2003. (2003 ; Berlin, Germany). First International XML Database Symposium, 8 September 2003 / Edited by Rainer Unland. – Berlin : Springer, 2003. – 293 p.
5. «Empirical Software Engineering Issues Critical Assessment and Future Directions», International Workshop (2006; Dagstuhl Castle, Germany), 26-30 June 2006 / Edited by R. Selby. – Berlin : Springer, 2007. – 209 p.
6. «Fundamental Approaches to Software Engineering», ETAPS 2004 (2004 ; Barcelona, Spain). 7th International Conference, Held as Part of the Joint European Conferences on Theory and Practice of Software, March 29 –April 2 2004 / Edited by Tiziana Margaria-Steffen. – Berlin : Springer, 2005. – 404 p.
7. «Information Systems Development», Thirteenth International Conference on Information Systems Development Advances in Theory, Practice and Education (2004 ; Vilnius, Lithuania). 9–11 September 2004 / Edited by Olegas Vasilecas. – New York : Springer, 2005. – 550 p.
8. «Programming Languages and Systems», APLAS 2008 (2008 ; Bangalore, India). 6th Asian Symposium, 9-11 December 2008 / Edited by G. Ramalingam. – Berlin : Springer, 2008. – 350 p.
9. «Web Information Systems Engineering», International Workshops (September 1–4, 2008 ; Auckland, New Zealand). 1–4 September 2008 / Edited by Markus Kirchberg. – Berlin : Springer, 2008. – 199 p.
10. Abmann U. Invasive Software Composition. / U. Abmann. – Germany: Springer, 2003. – 334 p.
11. Andersson E. Software Engineering for Internet Applications / E. Andersson, P. Greenspun, A. Grumet. – London: MIT Press, 2006. – 411 p.
12. Arnowitz J. Effective Prototyping for Software Makers / J. Arnowitz, M. Arent, N. Berger. – Canada: Elsevier, 2007. – 625 p.
13. Aurum A.·Engineering and Managing Software Requirements / A. Aurum; edited by C. Wohlin. – Berlin: Springer, 2005. – 487 p.
14. Bass L. Software Architecture in Practice / L. Bass, P. Clements, R. Kazman. – 2nd edition. – London : Addison Wesley, 2003. – 560 p.
15. Bernus P. Handbook on Architectures of Information Systems / P. Bernus, K. Mertins, G. Schmidt. – Berlin : Springer, – 2006. – 869 p.
16. Boucher T. Design of Industrial Information Systems / T. Boucher, A. Yaçin – Canada : Elsevier, 2006. – 431 p.
17. Brandon D.M. Software Engineering for Modern Web Applications: Methodologies and Technologies / D.M. Brandon – Hershey: IGI Global, 2008. – 403 p.
18. Cassidy A. A Practical Guide to Information Systems Process Improvement / A. Cassidy, K. Guggenberger – Florida: CRC Press LLC, 2001. – 286 p.
19. Cecelja F. Manufacturing Information and Data Systems / F. Cecelja – London: Penton Press, 2002. – 201 p.
20. Construx. Software Development Best Practices [Электронный ресурс] / Официальный сайт Construx; Steve McConnell – Электрон. дан. – Bellevue : Construx, 2009 – . – Режим доступа : http://www.construx.com, свободный. – Загл. с экрана. – Яз. англ.
21. Ebert C. Best Practices in Software Measurement. How to Use Metrics to Improve Project and Process Performance / C. Ebert, R. Dumke, M. Bundschuh, A. Schmietendorf. – Berlin: Springer-Verlag, 2005. – 299 p.
22. Giorgini P. Agent-Oriented Methodologies / P. Giorgini. – Hershey: Idea Group Inc., 2005. – 429 p.
23. Goodman F.A. Defining and Deploying Software Processes / F.A. Goodman. – New York : Taylor & Francis Group, 2006. –246 p.
24. Handbook of Enterprise Integration / edited by M.H. Sherif – New York : CRC Press, 2010. – 730 p.

25. International Function Point Users Group. [Электронный ресурс] / Официальный сайт International Function Point Users Group; President Tom Cagley – Электрон. дан. – Princeton : IFPUG, 2009 – . – Режим доступа : http://www.ifpug.org, свободный. – Загл. с экрана. – Яз. англ.
26. Jalote P. An Integrated Approach to Software Engineering / P. Jalote. – New York : Springer, 2005. – 571 p.
27. Kaisler S.H. Software Paradigms / S.H. Kaisler. – New Jersey : John Wiley & Sons, Inc., 2005. – 458 p.
28. Khosrow-Pour M. Cases on Information Technology: Planning, Design and Implementation / M. Khosrow-Pour. – London : Idea Publishing, – 2006. – 385 p.
29. Kirn S. Multiagent Engineering Theory and Applications in Enterprises / S. Kirn, O. Herzog, P. Lockemann ; edited by O. Spaniol. – Berlin: Springer, 2006. – 617 p.
30. Krishnamurthy N. Building Software. A Practitioner's Guide / N. Krishnamurthy, A. Saran. – New York : Taylor & Francis Group, 2008. – 380 p.
31. Krogstie J. Conceptual Modeling in Information Systems Engineering / J. Krogstie, A.L. Opdahl ; edited by S. Brinkkemper. – Berlin: Springer, 2007. – 356 p.
32. Laird L.M. Software Measurement and Estimation. A Practical Approach / L.M. Laird, M.C. Brennan. – New Jersey: IEEE Computer Society, 2006. – 276 p.
33. Langer A.M. Analysis and Design of Information Systems / A.M. Langer. – London: Springer, 2008. – 437 p.
34. Lattanze A.J. Architecting Software Intensive Systems. A Practitioner's Guide / A.J. Lattanze. – London: Taylor&Francis Group, 2009. – 464 p.
35. Lauesen S. User Interface Design. A Software Engineering Perspective / S. Lauesen. – Harlow : Addison Wesley, 2005. – 623 p.
36. Meszaros G. xUnit Test Patterns. Refactoring Test Code / G. Meszaros. – Boston: Pearson Education, 2007. – 948 p.
37. Olivé A. Conceptual Modeling of Information Systems / A. Olivé. – Berlin : Springer, 2007. – 471 p.
38. Pressman R.S. Software Engineering. A Practitioners Approach / R.S. Pressman. – New York : McGraw-Hill, 2001. – 888 p.
39. Pugh K. Interface-Oriented Design / K. Pugh. – Texas: The Pragmatic Programmers LLC, 2006. – 220 p.
40. Rosenberg D. Use Case Driven Object Modeling with UML. Theory and Practice / D. Rosenberg, M. Stephens. - New York: Springer-Verlag, 2007. – 471 p.
41. Salzman H. Software by Design. Shaping Technology and the Workplace / H. Salzman, S.R. Rosenthal. – New York : Oxford, 1994. – 361 p.
42. Sametinger J. Software Engineering with Reusable Components / J. Sametinger. – Berlin: Springer-Verlag, 1997. – 285 p.
43. Schmidt M. Implementing the IEEE Software Engineering Standards / M. Schmidt. – Indianapolis : SAMS, 2000. – 255 p.
44. Shoval P. Functional and Object-Oriented Analysis and Design: An Integrated Methodology / P. Shoval. – Hershey : Idea Group Publishing, 2007. – 357 p.
45. Spolsky J. User Interface Design for Programmers / J. Spolsky. –New York: Springer-Verlag, 2001. – 101 p.
46. Szyperski C. Component Software. Beyond Object-Oriented Programming / C. Szyperski, D. Gruntz, S. Murer. – New York : Addison-Wesley, 2002. – 625 p.
47. Tang X. Web Content Delivery / X. Tang, J. Xu, S.T. Chanson. – New York : Springer, 2005. – 387 p.
48. Vasilecas O. Information Systems Development : Advances in Theory, Practice and Education / O. Vasilecas, A. Caplinskas et al. – New York : Springer, 2005. – 550 p.
49. Wang L. Modern Industrial Automation Software Design / L. Wang, K. Tan. – New Jersey : John Wiley & Sons, Inc., 2006. – 349 p.
50. Амблер С. Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. Библиотека программиста / С. Амблер. – СПб: Питер, 2005. – 412 с.
51. Арлоу Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, А. Нейштадт. – 2-е изд. – СПб: Символ-Плюс, 2007. – 624 с.
52. Бек К. Экстремальное программирование: разработка через тестирование / К. Бек. – СПб.: Питер, 2003. – 224 с.
53. Белладжио Д. Стратегия управления конфигурацией программного обеспечения с использованием IBM Rational ClearCase / Д. Белладжио, Т. Миллиган. – М.: ДМК Пресс, 2007. – 384 с.

54. Благодатских В.А. Стандартизация разработки программных средств / В.А. Благодатских, В.А. Волнин, К.Ф. Поскакалов ; под ред. О.С. Разумова. – М.: Финансы и статистика, 2005. – 288 с.
55. Брауде Э. Технология разработки программного обеспечения / Э. Брауде. – СПб.: Питер, 2004. – 655 с.
56. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч, Р. Максимчук, М. Энгл и др. – 3-е изд. – М.: ООО «И.Д. Вильямс», 2008. – 720 с.
57. Вендров А.М. Проектирование программного обеспечения экономических информационных систем / А.М. Вендров. – М.: Финансы и статистика, 2006. – 544 с.
58. Вигерс К. Разработка требований к программному обеспечению / К. Вигерс. – М. : Издательско-торговый дом «Русская редакция», 2004. – 576 с.
59. Влиссидес Д. Применение шаблонов проектирования. Дополнительные штрихи / Д. Влиссидес. – М.: Издательский дом «Вильямс», 2003. – 144 с.
60. Гагарина Л.Г. Технология разработки программного обеспечения: учебное пособие / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Виснадул ; под ред. Л.Г. Гагариной. – М.: ИД «Форум»: ИНФРА-М, 2008. – 400 с.
61. Гайдамакин Н.А. Автоматизированные информационные системы, базы и банки данных. Вводный курс : учебное пособие / Н.А. Гайдамакин. – М.: Гелиос АРВ, 2002. – 368 с.
62. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон. – СПб.: Питер, 2008. – 366 с.
63. ГОСТ 19.003-80. Схемы алгоритмов и программ. Обозначения условные графические. – Введ. 1981–07–01. – М. : Изд-во стандартов, 1982. – 4 с.
64. ГОСТ 2.104-2006. Единая система конструкторской документации. Основные надписи. – Введ. 2006–02–28. – М. : Изд-во стандартов, 2006. – 18 с.
65. ГОСТ 2.105-95. Общие требования к текстовым документам. – Введ. 1996–07–01. – М. : Изд-во стандартов, 1998. – 28 с.
66. ГОСТ 24.103-84. Автоматизированные системы управления. Основные положения. – Введ. 1985–07–01. – М. : Изд-во стандартов, 1985. – 2 с.
67. ГОСТ 24.104-85. Автоматизированные системы управления. Общие требования. – Введ. 1987–01–01. – М. : Изд-во стандартов, 1987. – 7 с.
68. ГОСТ 24.602-86. Состав и содержание работ по стадиям создания. – Введ. 1988–01–01. – М. : Изд-во стандартов, 1988. – 6 с.
69. ГОСТ 34.003-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения. – Введ. 1992–01–01. – М. : Изд-во стандартов, 1992. – 14 с.
70. ГОСТ 34.601-90. Автоматизированные системы. Стадии создания. – Введ. 1992–01–01. – М. : Изд-во стандартов, 1992. – 3 с.
71. ГОСТ 34.602-89. Техническое задание на создание автоматизированной системы. – Введ. 1990–01–01. – М. : Изд-во стандартов, 1990. – 6 с.
72. ГОСТ Р 7.0.5-2008. Библиографическая ссылка. Общие требования и правила составления. – Введ. 2009–01–01. – М. : Изд-во стандартов, 2009. – 22 с.
73. ГОСТ Р ИСО/МЭК 12207-99. Информационная технология. Процессы жизненного цикла программных средств. – Введ. 1999–12–23. – М. : Изд-во стандартов, 2000. – 84 с.
74. Гранд М. Шаблоны проектирования в Java / М. Гранд. – М.: Новое знание, 2004. – 559 с.
75. Зельцер С.Р. Проектирование автоматизированных систем обработки информации и управления. Часть II. Общие вопросы проектирования: Учебное пособие / С.Р. Зельцер. НФИ КемГУ. – Новокузнецк, 2003. – 121 с.
76. Иванова Г. С. Технология программирования : учебник для вузов / Г. С. Иванова. – М.: Изд-во МГТУ им. Баумана, 2002. – 320 с.
77. Избачков Ю.С. Информационные системы : учебник для вузов. / Ю.С. Избачков, В.Н. Петров. – СПб.: Питер, 2006. – 656 с.
78. Ковшов А.Н. Информационная поддержка жизненного цикла изделий машиностроения : принципы, системы и технологии CALS / ИПИ : учеб. пособие для студ. высш. учеб. заведений / А.Н. Ковшов, Ю.Ф. Назаров, И.М. Ибрагимов и др. – М.: Издательский дом «Академия», – 2007. – 304 с.
79. Константайн Л. Разработка программного обеспечения / Л. Константайн, Л. Локвуд. – СПб.: Питер, 2004. – 592 с.

80. Ларман К. Применение UML и шаблонов проектирования / К. Ларман. – М.: Издательский дом «Вильямс», 2004. – 624 с.
81. Леффиигуэлл Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход / Д. Леффиигуэлл, Д. Уидриг. – М.: Издательский дом «Вильямс», 2002. – 448 с.
82. Липаев В.В. Программная инженерия. Методологические основы / В.В. Липаев. – М.: «ТЕИС», 2006. – 610 с.
83. Макконнелл С. Совершенный код. Мастер-класс / С. Макконнелл. – СПб.: Питер, 2005. – 896 с.
84. Маклаков С.В. Создание информационных систем с AllFusion Modeling Suite / С.В. Маклаков. – М.: ДИАЛОГ-МИФИ, 2003. – 432 с.
85. Мандел Т. Разработка пользовательского интерфейса / Т. Мандел. – М.: ДМК Пресс, 2001. – 416 с.
86. Мацяшек Л.А. Анализ и проектирование информационных систем с помощью UML 2.0 / Л.А. Мацяшек. – 3-е изд. – М. : ООО «И.Д. Вильямс», 2008. – 816 с.
87. Мыльник В.В. Исследование систем управления: Учебное пособие для вузов / В.В. Мыльник, Б.П. Титаренко, В.А. Волочиенко. – М.: Академический Проект; Екатеринбург: Деловая книга, – 2003. – 352 с.
88. Мюллер Р.Дж. Базы данных и UML. Проектирование / Р.Дж. Мюллер. – М.: «ЛОРИ», 2002. – 432 с.
89. Орлов С. Технологии разработки программного обеспечения : учебник / С. Орлов. – СПб. : Питер, 2002. – 464 с.
90. Перегудов Ф.И. Основы системного анализа / Ф.И. Перегудов, Ф.П. Тарасенко. – Томск: Изд-во НТЛ, 2001. – 396 с.
91. Проектирование информационных систем : курс лекций : учебное пособие для студентов вузов, обучающихся по специальностям в области информ. технологий / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. – М. : Интернет-Ун-т Информ. технологий, 2005. – 304 с.
92. Р 50.1.028-2001. Информационные технологии поддержки жизненного цикла продукции. Методология функционального моделирования. – Введ. 2001–07–02. – М. : ИПК Издательство стандартов, 2001. – 53 с.
93. Ройс У. Управление проектами по созданию программного обеспечения. Унифицированный подход / У. Ройс. – М. : Издательство «ЛОРИ», 1998. – 431 с.
94. Соммервилл И. Инженерия программного обеспечения / И. Соммервилл. – М. : Издательский дом «Вильямс», 2002. – 624 с.
95. Стелтинг С. Применение шаблонов Java. Библиотека профессионала / С. Стерлинг, О. Маасен. – М.: Издательский дом «Вильямс», 2002. – 576 с.
96. Тарасенко Ф.П. Прикладной системный анализ (Наука и искусство решения проблем) / Ф.П. Тарасенко. – Томск: Изд-во Том. ун-та, 2004. – 186 с.
97. Управляющие вычислительные комплексы: Учеб. пособие / Под ред. Н. Л. Прохорова. – 3-е изд. – М.: Финансы и статистика, 2003. – 352 с.
98. Фаулер М. Архитектура корпоративных программных приложений / М. Фаулер. – М.: «Вильямс», 2006. – 544 с.
99. Халл Э. Разработка и управление требованиями. Практическое руководство пользователя / Э. Халл, К. Джексон, Д. Дик. – Оксфорд: Springer, 2005. – 240 с.
100. Хетагуров Я.А. Проектирования автоматизированных систем обработки информации и управления (АСОИУ): Ученик / Я.А. Хетагуров. – М.: Высш. шк., 2006. – 223 с.
101. Хоп Г. Шаблоны интеграции корпоративных приложений / Г. Хоп, Б. Вульф. – М.: ООО «И.Д. Вильямс», 2007. – 672 с.
102. Шаврин О.И. Как формировать выводы по диссертации и составлять заключение диссертационного совета / О.И. Шаврин. – Ижевск: Изд-во ИжГТУ, 2002. – 28 с.
103. Якобсон А. Унифицированный процесс разработки программного обеспечения / А. Якобсон, Г. Буч, Дж. Рамбо. – СПб.: Питер, 2002. – 496 с.

ПРИЛОЖЕНИЕ А

(справочное)

ОБРАЗЕЦ ТИТУЛЬНОГО ЛИСТА ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

Министерство образования и науки Российской Федерации
Государственное образовательное учреждение высшего профессионального образования
Югорский государственный университет
Институт прикладной математики, информатики и управления
Кафедра «Автоматизированные системы обработки информации и управления»

Допущен(а) к защите

_____
(дата)

_____
(подпись)

КУРСОВОЙ ПРОЕКТ

по дисциплине: «Проектирование автоматизированных систем обработки информации и управления»
(специальность 230102 – Автоматизированные системы обработки информации и управления)

на тему: _____

Группа _____

Студент: ФИО студента _____
(подпись)

Руководитель: ФИО руководителя _____
(подпись)

Отметка о защите: _____

«_____» _____

Ханты-Мансийск 200_ г.

ПРИЛОЖЕНИЕ Б
(справочное)

ОБРАЗЕЦ ОФОРМЛЕНИЯ ЗАДАНИЯ НА КУРСОВОЙ ПРОЕКТ

“УТВЕРЖДАЮ”
_____ ФИО руководителя
“_____”_____ 200_г.

ЗАДАНИЕ ПО КУРСОВОМУ ПРОЕКТУ СТУДЕНТА

_____
(фамилия, имя, отчество)

1. Тема проекта: проектирование информационной системы учебно-методических материалов
_____

2. Срок сдачи студентом законченного проекта на рецензирование: _____
на защиту: _____

3. Исходные данные к проекту (организация, отрасль знаний, назначение проектируемой системы, основные требования):
информационная система учебно-методических материалов (УММ – книги, статьи, учебные пособия и т.д.) предназначена для: (i) обеспечения доступа во внутренней сети организации к следующей информации об УММ, хранящейся в базе данных: ФИО автора, название, год издания, количество страниц, аннотация; (ii) организации хранения файлов УММ. Пользователь системы с помощью web-интерфейса должен иметь возможность: (i) проводить поиск в базе данных по полям «автор», «название-аннотация»; (ii) копировать на носитель файлы найденных УММ; (iii) загружать в каталог сервера файлы УММ, отсутствующие в БД, для последующей их обработки администратором. Администратор системы с помощью web-интерфейса должен иметь возможность: (i) заносить записи в базу данных, проводить поиск, редактировать их и удалять; (ii) размещать файлы, загруженные пользователем, в соответствующий каталог файловой системы сервера; (iii) регистрировать новых пользователей в системе. Особое внимание при проектировании системы необходимо уделить защите файлов УММ и информации БД от повреждения вследствие сбоя и/или несанкционированного доступа.

4. Содержание пояснительной записки по курсовому проекту (перечень подлежащих разработке вопросов):
пояснительная записка по курсовому проекту включает в свой состав семь глав: (i) введение, (ii) проведение обследования, (iii) формирование требований, (iv) разработка концепции, (v) техническое задание, (vi) эскизный проект, (vii) заключение.
Приложение к пояснительной записке включает: план управления конфигурациями, план контроля качества программного обеспечения, план управления программным проектом, спецификация требований к программному обеспечению, техническое задание, проектная документация, отчет о проведенном инспектировании, диаграммы IDEF0(IDEF3), DFD, (ERD), UML, блок-схема алгоритма функционирования метода поиска УММ.
Индивидуальный вопрос, требующий специальной проработки: моделирование предметной области.
Функция системы, реализация которой должна быть описана в пояснительной записке: загрузка в каталог сервера файлы УММ, отсутствующие в БД, для последующей их обработки администратором.

5. Студент-инспектор: ФИО _____
(подпись)

6. Руководитель проекта _____
(подпись, дата)

7. Задание принял к исполнению _____
(подпись, дата)

ПРИЛОЖЕНИЕ В
(справочное)

ПРИМЕР ОФОРМЛЕНИЯ АННОТАЦИИ

АННОТАЦИЯ

					КП-115101-АСОИУ-09 *(№ зачетной книжки - выпускающая кафедра – год)*			
Изм	*Лист*	*№ документа*	*Подпись*	*Дата*				
Разраб.		*Петров А.И.*			[Наименование проектируемой системы]	*Литера*	*Лист*	*Листов*
Провер.		*Иванов А.А.*				*У*	*3*	*63*
Н.контр.		*Иванов А.А.*				*ИПМИиУ-ЮГУ*		
Утв.		*Иванов А.А.*						

ПРИЛОЖЕНИЕ Г

(справочное)

ПРИМЕР ОФОРМЛЕНИЯ СОДЕРЖАНИЯ

СОДЕРЖАНИЕ

ПРИЛОЖЕНИЕ Д

(справочное)

ПРИМЕР ОФОРМЛЕНИЯ КОМПЛЕКТА SADT-ДИАГРАММ

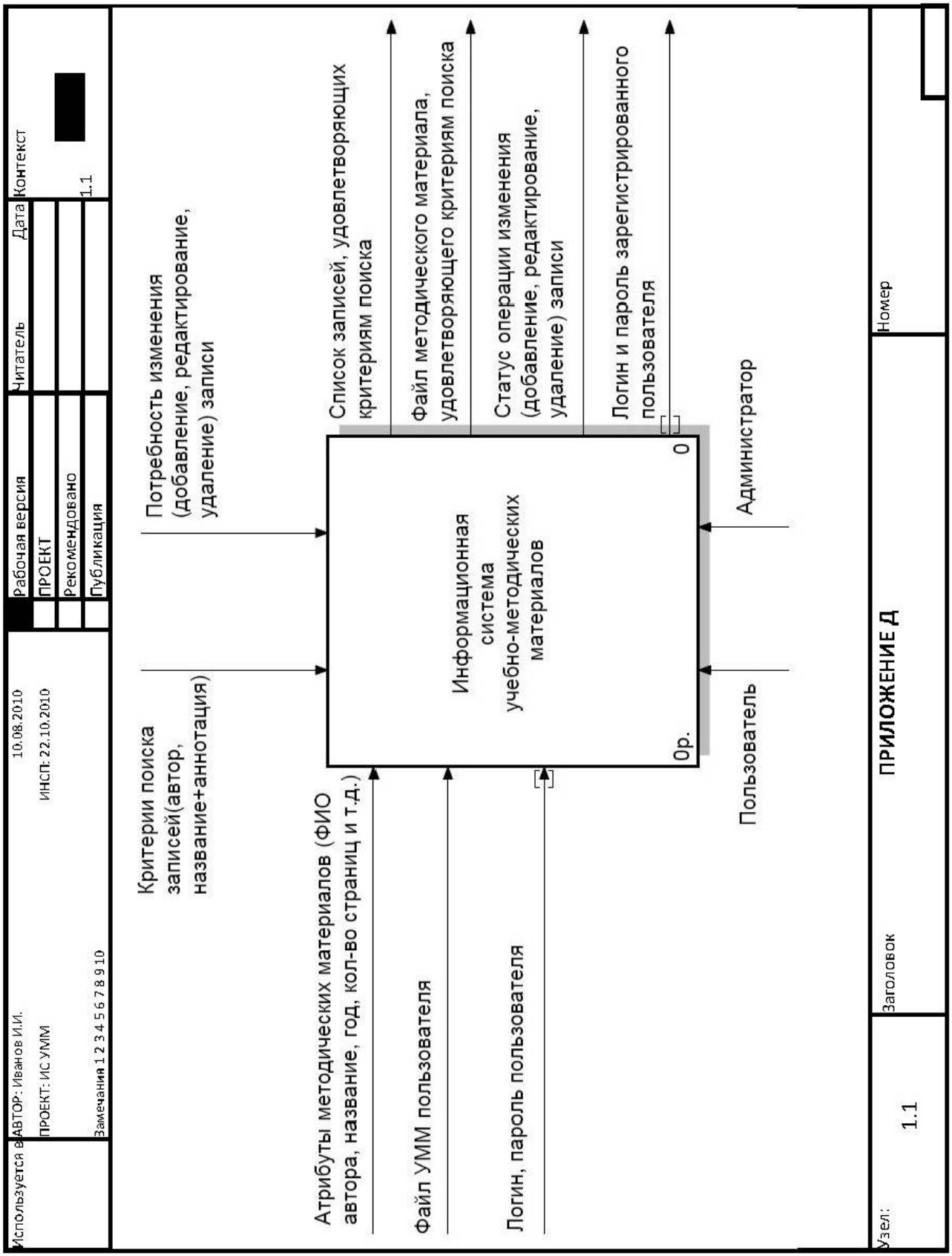

Рис. Д.1 Контекстная диаграмма IDEF0

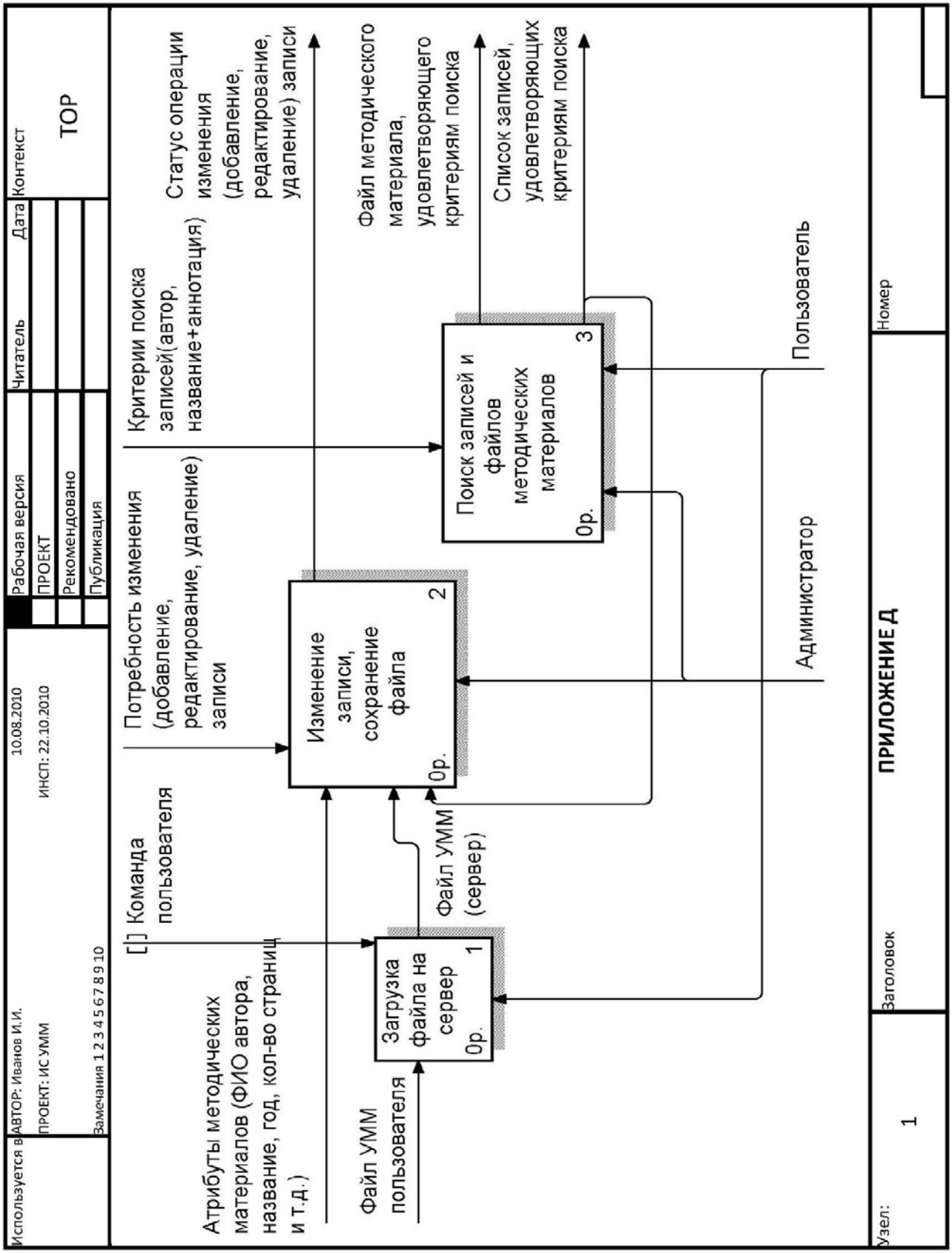

Рис. Д.2 Декомпозиция контекстной диаграммы IDEF0

ПРИЛОЖЕНИЕ Е

(справочное)

IEEE 828-1990 ПЛАН УПРАВЛЕНИЯ КОНФИГУРАЦИЯМИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (SOFTWARE CONFIGURATION MANAGEMENT PLAN – SCMP)

Утверждаю

_____Дата_____

01.06.09 А. Мелихов: Создание первой версии
10.06.09 А. Мелихов: Рецензирование
18.06.09 А. Мелихов: Расширен раздел 5.2
18.07.09 А. Мелихов: Проверка для выпуска
30.07.09 А. Мелихов: Окончательное форматирование
02.08.09 А. Мелихов: Выпуск

1. Введение

Данный план управления конфигурациями ПО (SCMP) описывает, как ведется работа с артефактами информационно-справочной системы (ИСС).

1.1. Сокращения

CI (Configuration Item) – элемент конфигурации – любой элемент, отслеживаемый системой управления конфигурациями.
CM (Configuration Management) – управление конфигурациями – процесс поддержки релевантных версий артефактов проекта.
SCMP (Software Configuration Management Plan) – данный документ.
SRS (Software Requirements Specification) – спецификация требований к программному обеспечению.
SDD (Software Design Document) – проектная документация программного обеспечения.
STD (Software Test Documentation по ANSI/IEEE 829-1983) – документация по тестированию программного обеспечения.
SPMP (Software Project Management Plan) – план управления программным проектом.
CMM (Capability Maturity Model) – модель технологической зрелости организаций.

1.2. Термины

Утвержденный CI — CI, подписанный руководством проекта.
Артефакт — окончательный или промежуточный материал проекта (например, документ, исходный код, объектный код, результат теста).
Главный файл — специальным образом построенный файл для данного проекта, определяется в разделе 3.1.2.

2. Управление конфигурациями

2.1. Организация

[*Определите, как должно быть организовано управление конфигурациями. Укажите роли, но не имена конкретных исполнителей — для этого служат другие разделы.*]
Специальный инженер, выделяемый отделом контроля качества, будет назначен ведущим конфигурацию на все время проведения данного проекта.

2.2. Ответственность за управление конфигурациями

[*Определите ответственность каждой роли*]

2.2.1. Ведущий конфигурацию

[*Ведущий конфигурацию организовывает работу и контролирует ход ее выполнения*]

Ведущий конфигурацию:

- отвечает за организацию и управление конфигурациями;
- обсуждает планы управления конфигурациями с командой разработчиков до того, как эти планы вводятся в действие;
- поддерживает данный документ;
- отвечает за установку и сопровождение инструментов управления конфигурациями, определенных в разделе 2.3.
- отвечает за настройку, сопровождение и резервное копирование используемых инструментов управления конфигурациями, а также разрабатывает план действий на случай, если используемые инструменты окажутся неподдерживаемыми (например, по вине поставщика).

Дополнительные обязанности ведущего конфигурацию описаны в разделах 3.3, 3.4, 3.5 и 3.6.

2.2.2. Лидер проекта

Лидер проекта и руководитель проекта могут выполнять функции ведущего конфигурацию только в исключительных обстоятельствах. Они обязаны знать все соответствующие средства доступа к документам во время проведения проекта. Лидер проекта обязан проверить, что архивирование данных ведется в соответствии с инструкций, упомянутой в разделе 2.3.

Дополнительные обязанности лидера проекта описаны в разделах 3.3 и 3.4.

2.2.3. Разработчики

Каждый разработчик обязан выполнять правила управления конфигурациями, опубликованные ведущим конфигурацию. Разработчики также обязаны следовать документу 56789 «Должностные обязанности инженеров». Дополнительные обязанности разработчика описаны в разделе 3.

2.3. Применяемые политики, директивы и процедуры

[*Такая деятельность, как управление конфигурациями, обычно осуществляется в соответствии с общей политикой организации. Студенты должны идентифицировать и перечислить в этом разделе соответствующие правила и инструкции. В частности, необходимо упомянуть систему управления конфигурациями (название SuperCMTool дано для примера), выбор которой был сделан на страницах пояснительной записки.*]

1. Управление конфигурациями для данного проекта должно осуществляться в соответствии с указаниями по управлению конфигурациями, изложенными в корпоративном документе 7890 версии 6 от 15.08.2008.
2. В соответствии с политикой улучшения процесса разработки требуется проводить обзорные совещания по ходу и в конце проекта. Результаты самооценки должны быть отправлены управляющему, ответственному за улучшение процесса, не позднее трех недель после того, как состоялось совещание. Все разделы, в которых указаны возможные улучшения, должны содержать соответствующий материал и конкретные примеры.
3. Все текущие и предшествующие версии CI должны сохраняться.
4. К главному файлу (определен в разделе 3.1.2) имеет доступ только ведущий конфигурацию, а в его отсутствие — начальник отдела.
5. Пароли управления конфигурациями должны меняться в соответствии с принятыми корпоративными правилами безопасности со следующим добавлением: никакой пароль не должен изменяться, пока лидер проекта, руководитель проекта и ответственный за качество не оповещены об изменении и не подтвердили получение оповещения.
6. Лидер проекта и начальник отдела должны всегда иметь полный доступ ко всем документам, которые затрагивает управление конфигурациями. Каждые две недели лидер проекта должен предоставлять форму http://localhost/division3.accessVerification своему начальнику для верификации прав доступа.
7. В настоящем проекте будет использоваться средство SuperCMTool версии 3.4, поставляемое компанией SuperCMTool.
8. Архивация должна производиться в соответствии с корпоративной инструкцией №12345

3. Виды деятельности

3.1. Определение конфигурации

[*В этом разделе определяется, как создаются элементы конфигурации (CI) и как им назначаются имена.*]

3.1.1. Определение элементов конфигурации

Лидер проекта несет ответственность за определение всех элементов конфигурации. Разработчики, желающие определить новый CI, должны получить согласие лидера проекта по электронной почте или иным способом. Если лидер проекта недоступен в течение более чем одного рабочего дня, ведущий конфигурацию имеет право включить в конфигурацию предлагаемый элемент.

3.1.2. Именование элементов конфигурации

Ведущий конфигурацию несет ответственность за маркировку всех CI. Соглашение об именах следующее.

Корневой каталог: IRS (Informational Reference System).

Вложенные каталоги: SRS или SDD или...

Файл с именем N_N_N.xxx соответствует версии N.N.N документа.

Например, версия SRS 2.4.8 имеет имя IRS/SRS/2_4_8.txt.

Главный файл с именем Master находится в корневом каталоге и содержит информацию о текущем состоянии и предыдущих состояниях проекта. Например, файл Master может включать такую инфор-

мацию: текущая версия проекта 3.7.1. - включает версию 2.4.8 SRS и версию 1.4 SDD; предыдущая версия проекта 3.6.11. - включает версию 2.4.8 SRS и версию 1.3 SDD.

3.1.3. Получение элементов конфигурации

[*Составляя данный раздел, примите во внимание наиболее стрессовую часть проекта, то есть фазу реализации, когда несколько человек будут работать параллельно.*]

Для модификации CI разработчик должен получить CI с помощью процедуры checkout инструмента SuperCMTool. Эта информация запоминается и предоставляется другим пользователям, желающим взять данный CI на модификацию. Любой желающий получить данный CI должен договориться с текущим владельцем CI о передаче средствами SuperCMTool. Ни при каких обстоятельствах разработчики не должны передавать друг другу CI прямо, в обход SuperCMTool. Для любого разработчика любой CI должен быть доступен на чтение в любое время.

3.2. Контроль конфигурации

[*Данный раздел устанавливает процесс внесения изменений в элементы конфигурации.*]

3.2.1. Запрос на изменения

Назначается инспектор для каждого участника команды. Прежде чем сделать запрос на изменение, разработчик обязан получить одобрение данного предложения по изменению от инспектирующей команды или, если последнее невозможно, от своего инспектора. Чтобы сделать запрос на изменение, необходимо предоставить форму http://localhost/ultracorp.division3.Encounter.submitCI ведущему конфигурацию и лидеру проекта вместе с исходным CI и измененным CI.

3.2.2. Оценка изменений

Лидер проекта или его заместитель оценивают все запросы на изменения. Лидер проекта должен также указать стандарты качества, которые необходимо учесть при внесении изменения.

3.2.3. Одобрение или отклонение изменений

Запрос на изменение должен быть одобрен лидером проекта. Если лидер проекта не имеет возможности это сделать в течение трех рабочих дней, то право одобрения запроса на изменение переходит к ведущему конфигурацию.

3.2.4. Реализация изменений

Как только CI включается в конфигурацию, на ведущего конфигурацию возлагается ответственность за тестирование и интеграцию изменений. Это должно выполняться в соответствии с правилами регрессионного тестирования, описанного в документации по тестированию программного обеспечения (STD). В частности, ведущий конфигурацию должен координировать сборку версии для тестирования.

Выпуск версий должен утверждаться лидером проекта или руководителем проекта, если лидер отсутствует.

3.3. Определение статуса конфигурации

Ведущий конфигурацию обязан обновлять информацию о конфигурации на странице http://localhost/uLtracorp.division3/Encounter/Configuration не реже раза в неделю. Для этого достаточно опубликовать итоговый отчет инструмента SuperCMTool.

3.4. Аудиты и обзоры конфигурации

Руководитель проекта должен запланировать выполнение обзоров конфигурации ведущим конфигурацию не реже, чем раз в две недели, обычно в качестве одного из пунктов повестки дня еженедельных собраний команды. Ведущий конфигурацию должен сделать обзор состояния конфигурации и предложить детальные процедуры управления конфигурациями на фазах кодирования и интеграции.

3.5. Управление интерфейсом

Система управления конфигурациями должна иметь интерфейс с веб-сайтом проекта. Этим интерфейсом управляет ведущий конфигурацию.

3.6. Контроль поставщиков и субподрядчиков

Ведущий конфигурацию должен отслеживать обновления и сообщения об ошибках в инструменте SuperCMTool. У него должен быть план действий на случай, если поддержка SuperCMTool будет прекращена. Этот план должен быть представлен лидеру проекта в течение месяца после начала проекта.

4. Расписание

[*План мероприятий по управлению конфигурациями можно определить в данном разделе или в общем плане в SPMP. В последнем случае данный раздел должен не дублировать SPMP, а содержать только ссылку.*]

5. Ресурсы

Ведущему конфигурацию для выполнения своих обязанностей требуется в среднем приблизительно 6 часов в неделю в первой половине проекта и 12 часов в неделю во второй половине проекта. Время, затрачиваемое другими разработчиками на управление конфигурациями, принято отдельно не учитывать.

6. Сопровождение

Ввиду важности наличия стабильного плана управления конфигурациями изменения в данном документе должны быть одобрены всей командой проекта.

Поскольку целью организации является достижение уровня 5 по CMM, ведущий конфигурацию при подготовке совещаний по улучшению процесса управления конфигурациями обязан сделать следующее:

- Оценить эффективность данного плана.
- Количественно оценить потери, вызванные дефектами данного плана.
- Оценить эффективность использования инструмента SuperCMTool.
- Изучить литературу по новым методам управления конфигурациями; количественно оценить выгоды и затраты на проведение улучшений.
- Изучить новые инструменты управления конфигурациями.
- Предложить конкретные улучшения текущего процесса управления конфигурациями.
- Перечислить выгоды от улучшений.
- Предоставить оценки стоимости эффекта введения улучшений.

Упорядочить предлагаемые улучшения по значению отношения выгоды-затраты.

ПРИЛОЖЕНИЕ Ж

(справочное)

IEEE 730-1989 ПЛАН КОНТРОЛЯ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (SOFTWARE QUALITY ASSURANCE PLAN – SQAP)

Утверждаю

_____Дата_____

17.01.09 А. Мелихов: Создание первой версии.

30.05.09 А. Мелихов: Рецензирование и добавление разделов от 7 до конца.

31.05.09 А. Мелихов: Интеграция и пересмотр содержания.

1. Цель

В этом документе описан план получения качественного продукта при выполнении проектирования информационно-справочной системы (ИСС).

2. Задействованные документы

См. раздел 4.2.

3. Управление

3.1. Организация

[*Этот раздел устанавливает, какие роли задействованы в процессе обеспечения качества. Фактическое распределение обязанностей дано в разделе 3.3.*]

Каждый участник команды отвечает за качество своей работы. Кроме того, на первые три итерации проекта назначается отдельный ответственный за качество. Ответственный за качество руководит всеми вопросами, связанными с обеспечением качества в проекте. Начиная с итерации 4 должна быть назначена команда инженеров по контролю качества, в которую должен войти ответственный за качество.

3.2. Задачи

Задачи по контролю качества в данном проекте включают в себя:

- документирование;
- обзорные собрания;
- верификацию (включая инспектирование);
- валидацию (прежде всего тестирование);
- виды деятельности, направленные на улучшение самого процесса обеспечения качества.

Эти задачи детализированы в данном документе.

3.3. Ответственность

[*Здесь не упоминаются имена сотрудников, поскольку они должны быть указаны в SPMP.*]

Ответственный за качество следит за тем, чтобы требования данного документа были выполнены, в том числе, чтобы были организованы обзорные собрания.

Лидер проекта отвечает за то, чтобы управление качеством действительно производилось.

Обязанности ответственного за требования и ответственного за проектирование описаны в разделе 6 данного документа.

4. Документация

4.1. Цель

В данном разделе определяется документация, используемая для обеспечения качества.

4.2. Минимальные требования к документации

[*В этом разделе перечисляются все документы проекта, поскольку именно проектная документация обеспечивает качество продукта.*]

Должны быть созданы следующие документы.

- SQAP – План контроля качества (данный документ).
- SCMP – План управления конфигурациями.
- SPMP – План управления программным проектом.
- SRS – Спецификация требований к программному обеспечению.
- SDD – проектная документация программного обеспечения.
- STD – документация по тестированию программного обеспечения

Помимо этих документов в исходном коде на Java будет использоваться Javadoc и, таким образом, можно будет генерировать документацию на уровне пакетов, классов и функций.

4.3. Прочее

План экспертизы программного обеспечения (SVVP) должен быть создан и поддерживаться независимо от SQAP.

5. Стандарты, практики, соглашения и метрики

5.1. Цель

[*Данный раздел описывает стандарты, практики, соглашения и метрики, используемые в проекте. Эти материалы призваны не только обеспечить качество проекта, но и получить количественные данные о самом процессе контроля качества.*]

5.2. Содержание

Стандарты.

Для ведения документации используются стандарты IEEE с соответствующими модификациями.

Практики.

1. При проектировании поощряется применять процедуры обеспечения качества.
2. Все артефакты проекта подлежат инспектированию и все они доступны после выпуска. Это достигается путем помещения артефактов в систему управления конфигурациями, обеспечивающей доступ к содержимому в любое время.
3. Для всех процессов должен быть проведен обзор возможности улучшения хотя бы раз, и результаты этого обзора в письменной форме должны быть направлены в лабораторию технологии программирования (раздел 6.2.10).

Соглашения.

Служба контроля качества организует трехчасовые курсы по изучению принятых соглашений ежемесячно. Присутствие на этих курсах оплачивается из накладных расходов организации.

Метрики.

Для каждого процесса и каждого документа должны измеряться по меньшей мере три показателя.

1. Время, затраченное сотрудником, на выполнение каждой подзадачи.
2. Самооценка качества по шкале от 1 до 10.
3. Количество дефектов на единицу объема (например, на тысячу строк кода).

Автор данного проекта ставит перед собой цель достичь следующих показателей качества своих продуктов.

[*Числа, использованные ниже, должны опираться на собранные ранее данные и отражать требуемый уровень соответствия ПЗ требованиям (см. табл. 8 на стр. 45).*]

Ограничения на число дефектов, обнаруживаемых в течение двух месяцев после поставки.

- Анализ требований: не более одного незначительного дефекта на сто детальных требований.
- Проектирование архитектуры: не более одного незначительного дефекта на пять диаграмм.
- Детальное проектирование: не более двух незначительных дефектов на тысячу строк.
- Кодирование: не более двух незначительных дефектов на тысячу строк кода, не считая комментариев.

Фактические данные по проекту должны быть представлены в приложении 1 к данному документу.

6. Обзоры и аудиты

6.1. Цель

[*Цель обзоров и аудитов состоит в том, чтобы привлечь внимание разработчиков к качеству приложения в ходе разработки. Обзоры проводятся на плановой регулярной основе. Объекты аудита выбираются случайным образом.*]

6.2. Минимальные требования

[*В больших проектах требуется полный список обзоров и аудитов, приведенный здесь. В рамках курсового проекта необходимо как минимум провести инспектирование требований и проектирования.*]

План проведения обзоров описан в SPMP. Суть обзоров указана ниже.

6.2.1. Обзор требований к программному обеспечению

Данный обзор проводится по всему документу с требованиями в присутствии всей команды. Обзор ведет лидер проекта. Предполагается, что требования не были инспектированы до данного обзора. Данный обзор не подменяет инспектирование требований. Ответственный за требования обязан проследить, чтобы инспектирование требований состоялось (см. SPMP).

6.2.2. Предварительный обзор проектных решений

Данный обзор альтернативных архитектурных решений выполняется всей командой. Обзор проводится лидером проекта или его заместителем. Подразумевается, что команда даст предложения, которые будут учтены в окончательном варианте проекта. Альтернативные архитектуры не должны инспектироваться до данного обзора. Ответственный за проектирование обязан проследить, чтобы данный обзор состоялся (см. SPMP).

6.2.3. Критический обзор проектных решений

Это инспектирование предложенной архитектуры в присутствии всей команды. Ответственный за проектирование обязан проследить, чтобы это инспектирование состоялось. Архитектура не должна инспектироваться до данного обзора. Если это возможно, архитектура должны быть разложена на составные части и по каждой из них должен быть проведен критический обзор проектных решений.

6.2.4. Обзор SVVP

Поскольку верификация и валидация должны выполняться независимой командой, в данном плане не предусматривается обзор SVVP.

6.2.5. Аудит функциональности

Лидер проекта перед поставкой продукта обязан организовать проверку того, что продукт соответствует SRS. Если необходимо сделать какие-то исключения, лидер проекта обязан получить предварительное разрешение на поставку от своего руководителя. Данные исключения должны быть доведены до сведения заказчика.

6.2.6. Аудит физических компонентов

До поставки ответственный за качество обязан организовать проверку того, что программное обеспечение и документация, предназначенные для поставки, физически включены в пакет поставки.

6.2.7. Аудиты процесса

Персонал проекта должен быть готов к внезапным аудитам своей работы. Аудит состоит в посещении рабочих мест командой, назначенной руководством отдела. Об аудитах следует предупреждать накануне. Предметом аудита является текущая работа отдельного разработчика или команды.

Поскольку организация движется в сторону CMM уровня 5, все рабочие материалы должны быть свободно доступны аудиторам и команде в любое время. Работа должна быть четко организована, так чтобы аудиты можно было проводить без предупреждения.

6.2.8. Обзор управления

Высшее руководство должно проводить обзор проекта в первую неделю каждого месяца. Лидер проекта ответственен за организацию таких обзоров.

6.2.9. Обзор SCMP

Ответственный за качество должен проводить обзор состояния управления конфигурациями не реже раза в месяц независимо от процедур, установленных в плане управления конфигурациями.

6.2.10. Заключительный обзор

Команда проекта должна проводить заключительный обзор после выполнения каждой фазы, для того чтобы накопить данные для последующих проектов. В заключительный обзор включается как обзор только что завершенной фазы, так и обзор самого процесса контроля качества. Ответственный за качество обязан составлять отчет об улучшении процесса по каждой фазе и предоставлять его руководству.

6.3. Инспектирование

[*В учебном проекте инспектирование проводится опекуном. В данном разделе описывается процесс инспектирования, который включает выполнение следующих шагов: (i) планирование; (ii) подготовка; (iii) проведение инспектирования; (iv) анализ причин появления дефектов; (v) коррекция; (vi) составление отчета*]

7. Тестирование

[*Здесь определяется, как будет осуществляться управление тестированием. Данный текст может ссылаться на STD, но не должен дублировать ее.*]

На первых трех итерациях все функции по контролю качества выполняет ответственный за качество. На последующих итерациях отдел контроля качества должен выделить команду по контролю качества и передать ей эти функции. Описание команды по контролю качества будет представлено. Команда проекта несет ответственность за тестирование отдельных методов и комбинаций методов в классе (модульное тестирование). Ответственный за качество (в дальнейшем — команда по контролю качества) ответственен за все остальные виды тестирования.

8. Отчеты о проблемах и коррекционная деятельность

[*В этом разделе объясняется, каким образом дефекты становятся известными, описываются и устраняются. Для отслеживания дефектов и их состояния, как правило, используется специальное программное обеспечение.*]

Форма отчета об обнаруженном дефекте в программном обеспечении, которую должна использовать команда проекта, представлена на рис. Ж.1. Для заполнения этой формы разработчики используют специальное приложение describeDefect. Номер дефекта определяется автоматически, и приложение гарантирует заполнение всех необходимых полей.

1. Номер дефекта:	2. Кто обнаружил:
3. Затрагивает документы:	
Затрагивает исходный код*	4. Пакет(ы) _____
5. Класс(ы) _____	6. Метод(ы) _____
7. Серьезность: _____	8. Тип: _____
9. Фаза появления**	
Треб □ Арх □ Дет. проект. □	Реализ. □ Интегр. □
10. Детальное описание: _____	
11. Решение:	12. Статус открыт/закрыт:
13. Подписи: описание и план инспектированы: _____	
14. Код решения и план тестирования инспектированы: _____	
15. Изменение одобрено: _____	

*Для дефектов в исходном коде **Самая ранняя фаза, на которой дефект уже существовал

Рис. Ж.1 Форма отчета об обнаруженном дефекте в программном обеспечении

Различаются следующие значения уровня серьезности дефектов:

- Серьезный дефект: наличие такого дефекта влечет невыполнение требований к программному обеспечению.
- Тривиальный дефект: такой дефект не влияет на выполнение или сопровождение приложения.
- Незначительный дефект: дефект, который не относится к двум предыдущим категориям.

В случае обнаружения тривиального дефекта разработчик не обязан создавать отчет о дефекте — достаточно послать сообщение по электронной почте тому, кто наиболее вероятно может устранить этот дефект.

В документах могут встречаться дефекты следующих типов: отсутствие материала, неясность, неоднозначность, неполнота, повтор (в том же документе или в другом) и противоречие.

В исходном коде могут встречаться следующие типы дефектов: синтаксические, логические, ошибки в данных (то есть переменная принимает недопустимое значение) и нарушения режима безопасности (то есть возможен недопустимый обход правил безопасности).

Ответственный за качество (а в дальнейшем команда по контролю качества) создает и поддерживает базу данных дефектов, в которой собраны отчеты о дефектах, описывающих все проблемы, несоответствия и аномалии проекта. Ответственный за качество должен обеспечить систематическую фиксацию найденных дефектов по указанной форме, направление их на рассмотрение и устранение. Направление дефектов на рассмотрение должно производиться в соответствии с SCMP.

9. Инструменты, технологии и методики

Методы управления качеством включают использование стандартов, прослеживание требований, верификацию проектирования, инспектирование программного обеспечения и верификацию формальными методами. Инструменты управления качеством включают в себя программы верификации программного обеспечения, списки контрольных вопросов, наклейки на носителях и штампы о приемке. Списки контрольных вопросов будут получены в лаборатории технологии программирования и адаптированы для проекта в соответствии с рекомендациями NASA.

Дополнительные инструменты, технологии и методы управления качеством описаны в SPMP.

10. Контроль программного кода

Методы и средства, используемые для поддержки, хранения и документирования версий компилируемого программного кода на всех фазах жизненного цикла, определены в SCMP. Команда по контролю качества должна убедиться, что процедуры, указанные в SCMP, действительно выполняются.

11. Контроль носителей

[*Здесь описываются правила обращения с дисками, лентами и т. п.*]

Команда по контролю качества проверяет, что носители, используемые для управления конфигурациями, установлены и протестированы. Факт проверки носителя командой по контролю качества удостоверяется с помощью штампа на наклейке. По результатам проверки носителей представляется отчет.

12. Контроль поставщиков

[*Данный раздел регулирует взаимоотношения с поставщиками программного и аппаратного обеспечения. Здесь описывается, как и кто осуществляет эти отношения.]*

Команда по контролю качества проверяет закупленные коммерческие программные продукты во время начальной проверки, удостоверившись, что имеются документы, подтверждающие целостность и версию продукта. Продукты валидируются в процессе установки и приемосдаточного тестирования.

13. Сбор, сопровождение и хранение протоколов

[*Здесь определяется, как физически будут храниться записи и кто за них отвечает. Сюда же относятся файлы, которые не находятся под управлением конфигурациями.*]

Записи, которые собирает и хранит команда по контролю качества, включают в себя:

- отчеты о выполненной работе;
- отчеты об обнаруженных аномалиях, которые составлены не по обычной форме отчетов о дефектах;
- ….

Помимо проверки того, что процедуры архивирования, предписанные SCMP, выполняются, команда по контролю качества должна архивировать свои собственные материалы не реже раза в неделю. Эти записи должны храниться и на фазах сопровождения и эксплуатации.

14. Обучение

Для участников команды разработчиков должен быть организован вводный курс по качеству на 4 часа. Курс должен включать описание метрик, которые будут использоваться, и лабораторное занятие по изучению инструментов для сбора метрик. Далее, должны быть организованы ежемесячные трехчасовые занятия для того, чтобы поддерживать у разработчиков знания инструментов и методов контроля качества. Разработчики могут не присутствовать на этих занятиях, если они получают оценку «отлично», пройдя тест на сайте по адресу GCI/monthly/SQA/quiz.

15. Управление рисками

Команда контроля качества должна стараться обнаружить факторы риска как можно раньше. Процедуры управления рисками описаны в разделе 3.3 SPMP.

ПРИЛОЖЕНИЕ И

(справочное)

IEEE 1058.1-1987 ПЛАН УПРАВЛЕНИЯ ПРОГРАММНЫМ ПРОЕКТОМ (SOFTWARE PROJECT MANAGEMENT PLAN – SPMP)

Утверждаю

_____Дата_____

05.01.09 А. Мелихов: Создание первой версии

02.02.09 А. Мелихов: Рецензирование и различные предложения по улучшению

16.05.09 А. Мелихов: Детализация плана-графика

29.05.09 А. Мелихов: Проверка для выпуска

1. Введение

1.1. Обзор проекта

Данный проект организован для разработки информационно-справочной системы (ИСС). Система будет разработана в несколько этапов, поскольку заказчиком специфицирование проекта будет выполняться последовательно с учетом результатов предыдущего этапа. Первые версии создаются в целях обучения, чтобы разработчики могли попрактиковаться в технологии разработки, и в качестве базы, на которой студенты могут создавать свои собственные проекты. Последующие версии, как ожидается, будут либо свободно распространяемыми, либо коммерческими продуктами.

1.2. Результирующие артефакты проекта

Следующие материалы должны быть поставлены в указанные сроки.

Версия 1 (прототип) с документацией — вторая неделя второго месяца.

Версия 2 с документацией — третья неделя пятого месяца.

Документация включает в себя SPMP, SQAP, SCMP, SRS, SDD (с использованием стандартов IEEE). Аббревиатуры определены в разделе 1.5.

1.3. Развитие SPMP

[*В этом разделе объясняется, как будет поддерживаться и развиваться данный документ.*]

Данный документ поддерживается лидером проекта. Лидер проекта должен поместить данный документ под управление конфигурациями и обязан поддерживать документ в актуальном состоянии, еженедельно внося необходимые изменения.

1.4. Ссылочные материалы

Все необходимые стандарты IEEE опубликованы в сборнике стандартов IEEE: Schmidt M. Implementing the IEEE Software Engineering Standards / M. Schmidt. – Indianapolis : SAMS, 2000. – 255 p.

1.5. Аббревиатуры

CI (Configuration Item) – элемент конфигурации.

CMM (Capability Maturity Model) – модель зрелости возможностей.

IEEE (Institute of Electrical and Electronics Engineers) – институт инженеров по электротехнике и радиоэлектронике.

QA (Quality Assurance) – контроль качества.

SEI (Software Engineering Institute) – институт технологий разработки программного обеспечения.

SCMP (Software Configuration Management Plan) – план управления конфигурациями программного обеспечения.

SPMP (Software Project Management Plan) – план управления программным проектом (данный документ).

SRS (Software Requirements Specification) – спецификация требований к программному обеспечению.

SDD (Software Design Document) – проектная документация программного обеспечения.

STP (Software Test Plan) – план тестирования программного обеспечения.

USDP (Unified Software Development Process) – унифицированный процесс разработки программного обеспечения.

2. Организация проекта

2.1. Модель процесса

Первые две версии проекта будут выполнены с использованием спирального процесса разработки, по одной итерации на версию. Итерации проводятся в соответствии с USDP. Согласно USDP, итерации классифицируются на начальные итерации, итерации проектирования, конструирования и

перехода. Первая итерация будет состоять только из анализа и планирования требований, вторая итерация будет первой в серии итераций проектирования. Это составит первую версию проекта. Количество последующих итераций и состав версии 2 будут определены после того, как заказчик ознакомится с первой версией.

2.2. Организационная структура

Организационная структура проекта в рамках ЮГУ представлена на рис. И.1. Проект организован как команда равных с назначением ролей. Роли следующие: лидер команды, ответственный за конфигурацию, ответственный за качество, ответственный за требования, ответственный за проектирование и ответственный за реализацию. Кроме того, имеется роль ответственного за связи с отделом маркетинга и с лабораторией технологий программирования. Все эти роли показаны на рис. И.2. В проекте будет проводиться инспектирование всей работы, как это определено в SQAP. Каждый участник команды будет проводить инспектирование работы других участников (см. рис. И.2).

Рис. И.1. Организационная структура

Рис. И.2. Организация проекта ИСС.

2.3. Организационные рамки и взаимосвязи

[*Указывают людей и организации, с которыми должна взаимодействовать команда.*]

Команда проекта должна взаимодействовать со следующими людьми и организациями: отдел разработки, отдел маркетинга, лаборатория информационных технологий, внешние эксперты и лаборатория технологии программирования.

2.4. Ответственность за проект

Ответственность участников проекта показана в табл. И.1. Ответственность за документ подразумевает следующее:

документ должен быть создан вовремя;

лидер команды определяет, кто пишет документ;

документ поддерживается в актуальном состоянии.

Таблица И.1.

Ответственность участников проекта

Участник	Лидер команды	Ответственный за конфигурацию	Ответственный за качество	Ответственный за требования	Ответственный за проектирование	Ответственный за реализацию
Отвечает за связь	Отдел разработки					
Отвечает за документ	SPMP	SCMP	SQAP	SRS	SDD	Листинги кода

3. Управляющий процесс

3.1. Цели и приоритеты

[*Устанавливается относительный приоритет расписания, бюджета и возможностей приложения. Возможности подразделяются на степень выполнения требований, уровень качества и пригодность для повторного использования. Повторное использование должно предусматриваться требованиями.*]

Высший приоритет имеет достижение заданного уровня качества. На втором месте по приоритетности стоит выполнение проекта в срок. Третий приоритет имеет выполнение максимального количества требований. Четвертый приоритет имеет создание классов, которые можно повторно использовать в других аналогичных проектах.

3.2. Допущения, зависимости и ограничения

[П*еречисляются предположения и допущения о внешних по отношению к проекту событиях, которые могут повлиять па проект.*]

Нет.

3.3 Управление рисками

[*Следует определить риски как конкретные неприятности, которые могут произойти. Нельзя ограничиваться общими фразами. Например, утверждение «недостаточные навыки программирования*

на Java» само по себе не описывает суть проблемы. Иногда проект можно успешно выполнить и с недостаточными навыками.]

Информация об идентификации и обработки рисков заносится в табл. И.2. Каждое совещание по проекту должно включать в повестку дня пункт «мозговой штурм с целью выявления рисков».

Риск № 1 - недостаточные навыки программирования на Java, отражает тот факт, что 40 % команды не имеют достаточного опыта программирования на Java. Принимая этот факт, следует признать, что данный фактор серьезно затормозит проект, но поскольку не все в команде должны программировать, ущерб оценивается числом 6. Стоимость устранения риска высока, поскольку нужно затратить много времени на обучение с отвлечением от работы, так что полагаем ее равной 8.

Риск № 2 …

Таблица И.2

Идентификация и обработка рисков

№ п/п	Название	Вероятность возникновения риска, 1-10 (1 - маловероятный)	Ущерб, 1-10 (1 - наименьший)	Стоимость устранения, 1-10 (1 - низкая)	Итоговый приоритет	План устранения (умень шения)	Ответственный за исправление	Дата завершения
1	недостаточные навыки программирования на Java	9	6	8	(11-9)*(11-6)*8=80	Изучение литературы: Java 2. Библиотека профессионала. Основы	А. Мелихов	09.09.09
2	…	…	…	…	…	…	…	…

3.4. Механизмы мониторинга и контроля

Вся команда будет встречаться на совещании в начале каждой фазы (определение требований, проектирование, реализация) каждой итерации. Должны проводиться еженедельные совещания по проекту по вторникам с десяти утра до полудня.

3.5. План расстановки кадров

Назначение ролей указано в табл. И.3. Каждый участник команды имеет дополнительные обязанности по резервированию и инспектированию (см. рис. И.2).

Таблица И.3

Назначение ролей участников проекта

ФИО	Лидер проекта	Ответственный за конфигурацию	Ответственный за качество	Ответственный за требования	Ответственный за проектирование	Ответственный за реализацию
А. Мелихов	X					X
А. Мелихов		X				
А. Мелихов			X			
А. Мелихов				X		
А. Мелихов					X	

Осуществляет связь с	Отдел разработки			Отдел маркетинга	Лаборатория информационных технологий и управления	

4. Технический процесс

[*В этом разделе описывается технология, используемая в проекте, но только в тех аспектах, которые не являются спецификацией требований.*]

SRS описывает некоторые аспекты требуемого технологического процесса. Здесь рассматриваются те аспекты, которые не установлены явно в SRS.

4.1. Методы, инструменты и технологии

В проекте для проектирования используется Rational Rose, а реализация ведется на языке Java. Во всех случаях используется объектно-ориентированный подход. Для документирования используется Javadoc (подробнее см. SRS). Используемая модель процесса описана в разделе 2.1.

4.2. Документация программного обеспечения

См. SQAP, раздел 4.2.

4.3. Функции сопровождения проекта

На все время проведения проекта будет привлечен на полставки технический специалист по поддержке инструментов.

5. Распределение работ, план-график и бюджет

5.1. Распределение работ

Распределение работ показано на рис. И.3. В нижней строке показано количество человеко-месяцев, приходящихся на данный месяц.

Рис. И.3. Распределение работ по проекту

5.2. Зависимости

[*В этом разделе показываются взаимозависимости различных работ. Данный раздел должен быть пересмотрен после проектирования.*]

5.3. Потребности в ресурсах

[*Проводится оценка стоимости разработки первых трех версий продукта.*]

В проекте будут заняты семь инженеров, один секретарь на полставки и один технический специалист на полставки.

Аппаратные ресурсы составят восемь компьютеров с процессором Intel Core 2 3,0 ГГц и операционной системой Windows Vista и системой программирования Symantec Visual Cafe версии 3.0. Каждый компьютер должен иметь не менее 2 Гбайт RAM и не менее 16 Гбайт свободного пространства на жестком диске.

5.4. Выделение бюджета и ресурсов

[*В этом разделе определяется, как расходуются выделенные средства. Смете предшествует оценка стоимости. По мере продвижения проекта оценка стоимости уточняется.*]

Оценка до начала анализа требований.

Данная оценка проведена тремя различными методами.

1. С использованием неформальной оценки сверху вниз на основе предыдущего опыта команды по аналогичным проектам.
2. С использованием оценки сверху вниз на основе данных по отрасли.
3. С использованием оценки функционального размера для двух известных функций и экстраполяцией результатов на весь проект.

Результаты приведены в табл. И.4 (все величины даны в тысячах строк исходного кода на языке Java).

Таблица И.4

Предварительная оценка размера приложения до анализа требований

Метод	Минимум	Максимум	Комментарий
1	7,5	170	
2	4,2	300	
3	11,4	46	1,9-2,3 для двух функций, умноженные на 6-20 по числу функций всего приложения

5.5. План-график

[*В настоящем разделе приводится итоговая версия плана-графика. Структура плана-графика должна соответствовать выбранной модели жизненного цикла проектируемой системы.*]

План график приведен на рис. И.4. См. также SQAP (приложение Ж), где приведен план-график мероприятий по контролю качества.

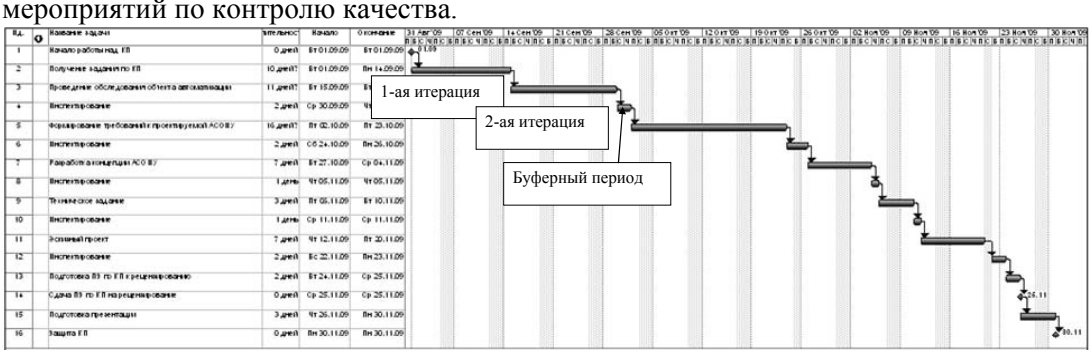

Рис. И.4. Диаграмма задач для проекта при условии фиксированной даты поставки

6. Дополнения

6.1. Указатель

6.2. Приложения

ПРИЛОЖЕНИЕ К

(справочное)

IEEE 830-1993 СПЕЦИФИКАЦИЯ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ (SOFTWARE REQUIREMENTS SPECIFICATION – SRS)

Утверждаю

_____Дата_____

02.02.09 А. Мелихов: исходный черновик
03.04.09 А. Мелихов: проверка технической точности; изменения текста
04.07.09 А. Мелихов: проверка всего документа, небольшие улучшения
06.15.09 А. Мелихов: перемещение вариантов использования в раздел 2.2
06.23.09 А. Мелихов: окончательное форматирование
06.23.09 А. Мелихов: выпуск

1. Введение

1.1. Цель

[*В разделе отражается цель данного документа в целом (а не цель программы)*]

Этот документ предоставляет все требования для проекта. Части 1 и 2 предназначены преимущественно для заказчиков приложения, но также будут интересны инженерам-разработчикам, разрабатывающим или поддерживающим его. Часть 3 предназначена в основном для разработчиков, но также представляет некоторый интерес и для заказчика.

1.2. Область применения

[*Аспекты программы, которые должен охватить этот документ*]

Этот документ охватывает требования к версии 0.01 информационно-справочной системы учебно-методических материалов (ИСС УММ).

1.3. Определения, термины и сокращения

C-требование – сводка требований к приложению, сформулированных в форме, понятной клиенту.
D-требование – сводка требований к приложению, сформулированных достаточно четко для использования программистами при проектировании и реализации. По возможности D-требования должны быть также понятны и клиенту

1.4. Ссылки

План управления конфигурациями программного обеспечения (SCMP) для ИСС, версия 1.0.
Архитектура программного обеспечения (SDD) для ИСС, версия 1.2.
План управления программным проектом (SPMP) для ИСС, версия 1.2.
План контроля качества (SQAP) для ИСС, версия 1.0.
План пользовательской документации (SUDP) для ИСС, версия 1.0.

1.5. Обзор

Обзор будет проведен в разделе 2.

2. Общее описание

[*В данном разделе проектируемая система описывается с точки зрения ее назначения в целом. Это описание необходимо сделать достаточно общим, чтобы оно несущественно изменялось в будущих версиях. Кроме того, здесь же перечисляются общесистемные и отраслевые требования к системам заданной класса.*]

Информационно-справочная система учебно-методических материалов (УММ) предназначена для обеспечения доступа во внутренней сети организации к следующей информации об УММ, хранящейся в базе данных сервера (ФИО автора, название, раздел знаний, год издания, количество страниц, аннотация), а также для организации доступа пользователей к файлам УММ, хранящимся на сервере. Пользователь системы с помощью web-интерфейса может проводить поиск в базе данных по полям «автор» или «название-аннотация» и копировать на свой носитель файлы найденных УММ из серверного хранилища. Кроме того, пользователь может загружать в специальный каталог сервера файлы УММ, отсутствующие в БД, для последующей их обработки администратором. Администратор системы с помощью web-интерфейса может: (i) проводить поиск записей, заносить новые записи в базу данных, редактировать их и удалять; (ii) размещать файлы, загруженные пользователем, в со-

ответствующий каталог сервера с одновременным созданием соответствующей записи в БД; (iii) регистрировать новых пользователей в системе с предоставлением им соответствующих прав.

2.1. Перспективы продукта

[*В данном разделе проектируемая система сравнивается с другими похожими или конкурирующими продуктами.*]

2.1.1. Концепции операций

[*Этот раздел дает общее представление о проектируемой системе. При этом используются те средства, которые в наибольшей степени для этого подходят: DFD, UML Use Case, UML Activity Diagram и т.д.*]

Контекстная диаграмма информационной системы УММ приведена на рисунке К.1. Входами системы являются: (i) атрибуты методических материалов, которые может использовать администратор при добавлении, редактировании, удалении записей из БД; (ii) файл УММ, отсутствующего в системе, который пользователь посредством web-интерфейса загружает в специальный каталог на сервере; (iii) логин и пароль пользователя, которые он вводит при авторизации в системе. Выходами системы являются: (i) список записей, удовлетворяющих критериям поиска и включающих в свой состав: ФИО автора, название УММ, год, количество страниц, область знаний, гиперссылку для скачивания файла УММ; (ii) файл УММ, удовлетворяющий критериям поиска, который пользователь может сохранить в указанный каталог, щелкнув по гиперссылке; (iii) статус операции (код ошибки) изменения записи в БД администратором; (iv) логин и пароль пользователя при регистрации в системе впервые. Так как критерии поиска (автор, название+аннотация) не изменяются системой в процессе функционирования, на рис. К.1 они представлены в виде потока управления. Кроме того, к управляющим входам относится необходимость в изменении той или иной записи, которая будет управлять процессом добавления, редактирования или удаления записи из БД. Осуществлять поиск, изменение записей, загрузку файлов УММ будут пользователи и администратор в соответствии с наделенными правами. Прочие механизмы (ПК, программное обеспечение и т.д.) являются тривиальными и на контекстной диаграмме не показаны.

Рис. К.1. Контекстная диаграмма информационной системы учебно-методических материалов

На рисунке К.2 представлена диаграмма декомпозиции контекстной диаграммы ИС УММ. Диаграмма декомпозиции включает в свой состав работу «изменение записи, сохранение файла», которая выполняется администратором и предназначена для добавления новой записи к БД с одновременной загрузкой файла УММ, а также для изменения имеющихся записей БД. В последнем случае необходимо определить изменяемую запись, проведя поиск с помощью работы «поиск записей и файлов методических материалов» (для этого на диаграмме (рисунок К.2) проведена обратная связь по входу между блоками 3 и 2). Как можно видеть из диаграммы декомпозиции пользователь имеет

возможность проводить поиск и загружать файл УММ на сервер. Изменять записи БД и помещать файлы УММ в соответствующий каталог системы может только *Администратор*.

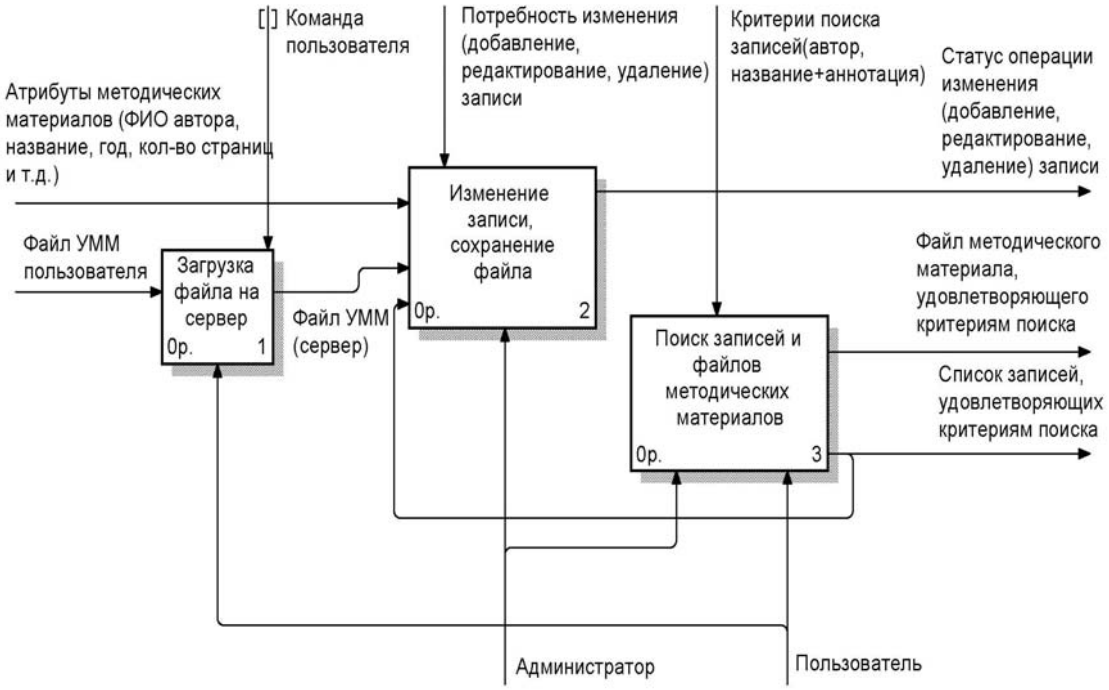

Рис. К.2. Диаграмма декомпозиции контекстной диаграммы ИС УММ

Дальнейшая декомпозиция контекстной диаграммы ИС УММ приведена в п. 1.2.2 пояснительной записки.

2.1.2. Концепции пользовательского интерфейса

[*В данном разделе размещается эскиз пользовательского интерфейса для общего представления о продукте. Итоговая версия пользовательского интерфейса описывается в разделе 3.1.1*

В настоящем проекте эскиз пользовательского интерфейса можно проработать и представить в виде фрагментов для каждого варианта использования (см. раздел 2.2) по отдельности, поскольку каждый вариант использования отвечает за отдельную функцию системы.]

2.1.2.1. Концепция пользовательского интерфейса для функции Ф.1 «Загрузить файл УММ на сервер»

Для загрузки файла УММ на сервер от пользователя понадобиться указать местоположение загружаемого файла на носителе. Для этого, как правило, используют поле для ввода пути к файлу с клавиатуры (рис. К.3) и командную кнопку для указания местоположения файла с помощью стандартного диалога операционной системы «открыть файл» (кнопка «Обзор…» на рис. К.3). Для передачи файла на сервер предусмотрена отдельная командная кнопка «Загрузить».

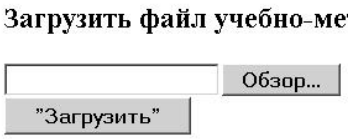

Рис. К.3. Эскиз пользовательского интерфейса функции Ф.1 «Загрузить файл УММ на сервер»

…

2.1.2.2. Концепция пользовательского интерфейса для функции администратора Ф.2 «Изменение записи, редактирование файла»

2.1.3. Аппаратные интерфейсы

[*Описание и анализ требований к аппаратным интерфейсам являются наиболее актуальными для систем реального времени (СРВ)*]

Нет.

2.1.4. Программные интерфейсы

[Проводится описание и анализ взаимодействия проектируемой системы с программным обеспечением сторонних разработчиков]

Согласно заданию на курсовой проект проектируемая система должна иметь web-интерфейс и клиент-серверную архитектуру. В этой связи для функционирования системы необходимо установить web-сервер, который при обращении пользователя к той или иной страницы через браузер будет обращаться к интерпретатору языка, на котором написаны скрипты программы, результаты выполнения скрипта в виде html-страницы будут возвращены в браузер пользователю (рис. К.4).

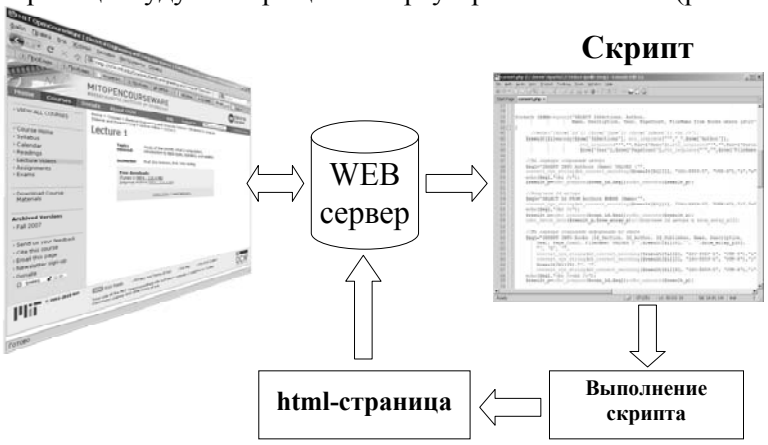

Рис. К.4. Программная структура проектируемой системы

Подробнее структура серверной части проектируемой системы будет описана в SDD.

2.1.5. Коммуникационные интерфейсы

[*Описание и анализ требований к коммуникационным интерфейсам являются наиболее актуальными для телекоммуникационных систем при разработке нестандартного программного обеспечения поддержки сетевых технологий. В большинстве корпоративных информационных систем используются стандартные коммуникационные интерфейсы и протоколы, которые не нуждаются в разработке дополнительного программного обеспечения для их поддержки.*]

Проектируемая информационная система предназначена для работы во внутренней сети организации, использующей стандартный стек протоколов сетевого уровня TCP/IP и протоколы HTTP, FTP прикладного уровня (в терминологии модели OSI).

2.1.6. Ограничения по памяти

Для функционирования ИС потребуется не менее 32 Мбайт свободной оперативной памяти и не менее 20 Мбайт свободного пространства на запоминающем устройстве. Объем свободного пространства, предназначенного для хранения файлов УММ, зависит от их количества. Необходимо учесть, что в среднем один файл УММ занимает около 7 Мб.

2.1.7. Операции

[*Перечисляются обычные и особенные операции, требуемые от пользователя.*]

2.1.8. Требования по адаптации

[*Требования по адаптации позволяют заложить при проектировании системы, а затем реализовать потенциальные потребности будущих пользователей: выбор языка интерфейса, цветовой схемы, подключения дополнительных модулей (plug-ins) и т.д.*]

В процессе эксплуатации системы может понадобиться введение дополнительных полей в БД, отсутствующих в ней изначально. С этой целью, для администратора должен быть предусмотрен соответствующий интерфейс.

2.2. Функции продукта

[*Это сводка сведений об основных функциях приложения: более подробная, чем в разделе 1.5; менее подробная, чем в разделе 3.]*

Анализ, проведенный на страницах пояснительной записки, позволил показать, что варианты использования являются наиболее подходящим способом определения общей функциональности проектируемой ИС. На рис. К.5 приведена первая (эскизная) версия диаграммы вариантов использования ИС УММ. Итоговую (уточненную) версию см. в п. 3.2. Ниже для каждого варианта использования приводится неформальное описание его функциональности.

2.2.1. Диаграммы вариантов использования

2.2.1.1. Вариант использования П.1 «Добавить файл УММ для обработки»

Для добавления УММ, отсутствующего в системе, *Читателю* необходимо загрузить файл УММ на сервер. Затем *Администратор* может открыть тот или иной файл для просмотра. Если содержание файла не соответствует тематике системы, *Администратор* удаляет файл. По окончании обработки файла УММ система автоматически посылает *Читателю* письмо с соответствующим уведомлением.

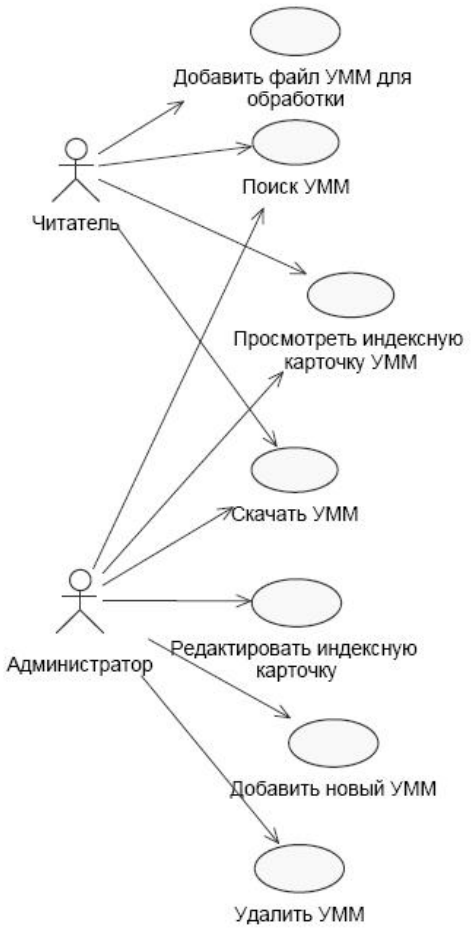

Рис. К.5. Диаграмма вариантов использования ИС УММ (эскиз)

2.2.1.2. Вариант использования П.2 «Поиск УММ»

…

2.3.1. Диаграммы последовательностей

[*Диаграммы последовательностей в ряде случаев интерпретируются как часть модели прецедентов, поскольку они обеспечивают визуализацию взаимодействия объектов при реализации прецедентов. В этой связи диаграммы последовательностей для каждого варианта использования строят в два этапа: (i) описание взаимодействия актеров с системой в целом как с черным ящиком (в разделе C-требования); (ii) описание взаимодействия объектов того или иного прецедента (в разделе D-требования).*]

На первом этапе построения диаграмм последовательностей будет рассмотрено взаимодействие актеров с системой в целом.

2.3.1.1. Диаграмма последовательностей варианта использования П.1 «Добавить файл УММ для обработки»

На диаграмме последовательностей (рис. К.6) отражено моделирование основного потока варианта использования «Добавить файл УММ для обработки» (см. п. 2.2.1.1) и, кроме того, показаны сообщения, которыми обмениваются объекты. На рис. К.6 каждое сообщение снабжено заметкой и в дополнительном комментарии не нуждается.

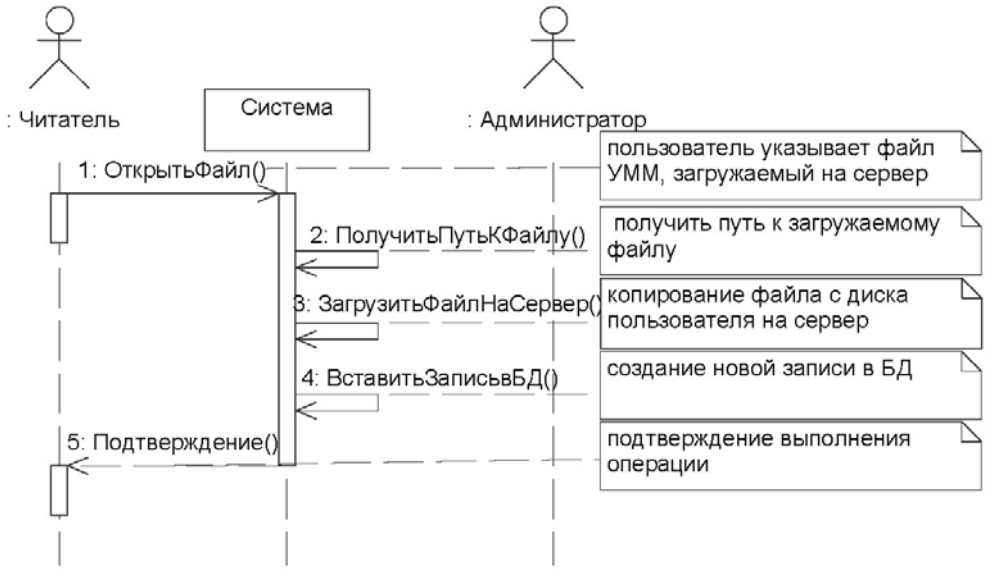

Рис. К.6. Диаграмма последовательностей для варианта использования П.1 «Добавить файл УММ для обработки»

2.2.2.2. Диаграмма последовательностей варианта использования П.2 «Поиск УММ»

…

2.2.3. Диаграммы классов

[*Необходимо описать назначение и взаимосвязь основных классов (классов анализа), полученных по результатам анализа предметной области. Уточненные диаграммы классов описываются и анализируются в разделе «D-требования»*]

2.2.3.1. Диаграмма классов варианта использования «Добавить файл УММ для обработки»

Для моделирования предметной области в отношении варианта использования П.1 была построена диаграмма классов, представленная на рис. К.7.

На рис. К.7 *Спецификация загрузок УММ* моделирует таблицу БД, в которую будут сохраняться данные (логин пользователя, дата, имя файла) о загружаемых на сервер файлах УММ. В *Спецификации УММ* для каждой УММ сохраняется атрибутивная информация (название УММ, год издания, кол-во страниц и т.д.), состав которой перечислен в задании на КП.

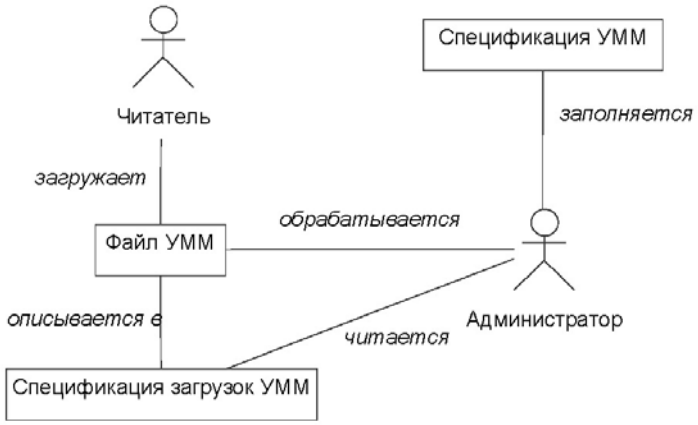

Рис. К.7. Диаграмма классов варианта использования П.1 «Добавить файл УММ для обработки»

2.2.3.2. Диаграмма классов варианта использования П.2 «Поиск УММ»

…

2.2.4. Диаграммы деятельности

[*Диаграмма деятельности отражает управленческий аспект моделирования процессов в системе, но в отличие от диаграммы последовательностей сосредотачивает внимание не на объектах, а на операциях. На стадии формирования С-требований для каждого варианта использования составляют упрощенную диаграмму деятельности, отражающую ключевые этапы выполнения того или иного прецедента.*]

2.3.1.1. Диаграмма деятельности варианта использования П.1 «Добавить файл УММ для обработки»

На диаграмме деятельности (рис. К.8) помимо основного (успешного) потока варианта использования П.1 показаны некоторые альтернативные сценарии. В частности, если загружаемый *Читателем* на сервер файл УММ имеет размер, превышающий установленное ограничение (*max*), то система блокирует загрузку файла и выдает соответствующее сообщение. Кроме того, если загрузка файла завершилась с ошибкой, система сообщает об этом пользователю и переходит на начальную страницу.

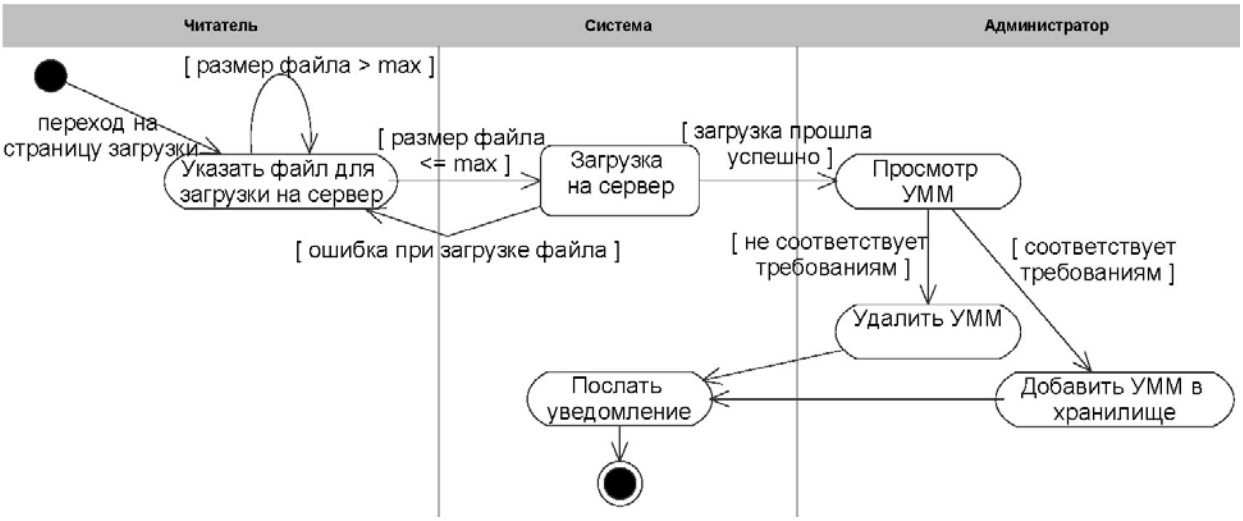

Рис. К.8. Диаграмма деятельности варианта использования П.1 «Добавить файл УММ для обработки»

2.3. Пользовательские характеристики

[*Покажите, какие люди будут типичными пользователями программы. Их уровень владения компьютером, а также психофизиологические особенности позволят сформулировать требования к проработке помощи, к детализации комментариев на каждом экране, необходимость использования мастеров, требования к дизайну экранов и т.д.*]

2.4. Ограничения

[*Ограничения на проектирование или реализацию описывают границы или условия того, как приложение должно быть задумано и разработано. Например, накладываются ограничения на инструменты и языки программирования ввиду сложившихся традиций организации, опыта программистов, совместимости и проч.*]

По настоянию заказчика проектирование и реализация проекта должна проводиться с использованием бесплатного (с открытым исходным кодом) программного обеспечения (Open Source). Анализ инструментальных платформ поддержки проектирования и разработки системы приведен в главе «Техническое задание» ПЗ.

Процедура предоставления доступа к авторским УММ регламентируется ст. 138 ГК РФ. В этой связи системой должен предоставляться доступ к УММ только во внутренней сети организации в учебных целях. Кроме того, по первому требованию автора или его представителя в случае несогласия на предоставления доступа к УММ автора, доступ к файлу УММ должен быть заблокирован администратором системы.

2.5. Предположения и зависимости

[*Могут быть сделаны любые допущения.*]

2.6. Распределение требований

Требования, описанные в разделах 1 и 2 этого документа называются «C-требованиями», в разделе 3 — «D-требованиями». Основной аудиторией C-требований является сообщество заказчиков, вторичной — разработчиков. Для D-требований ситуация обратная. Эти два уровня требований должны быть согласованными. Несогласованности должны быть отмечены отдельно как дефекты. В случае, когда требование сформулировано в C-требованиях и D-требованиях, приложение будет разрабатываться согласно D-требованиям, поскольку они более подробны.

3. Детальные требования

[*Детальные требования представляют полный список конкретных свойств и функциональности, которую должна иметь программа. Каждое детальное требование должно быть пронумеровано и отслеживаться в процессе разработки. D-требования должны быть согласованы с C-требованиями.*]

3.1. Требования к внешнему интерфейсу

3.1.1. Пользовательские интерфейсы

[*В данном разделе приводится и описывается итоговый вид графического пользовательского интерфейса (ГПИ), полученный на основе эскиза (п. 1.2.1.) и концепции ГПИ, представленной в разделе «Формирование требований к АСОИУ (ИМС)» пояснительной записки. При описании ГПИ необходимо в тексте настоящего раздела указать назначение каждого элемента*]

3.1.1.1. ГПИ варианта использования П.1 «Добавить файл УММ для обработки»

…

3.1.1.5. ГПИ варианта использования П.5 «Редактировать индексную карточку»

На рис. К.9 приведен ГПИ варианта использования «Редактировать индексную карточку», для построения которого был использован административный интерфейс фреймворка Django (см. п. 3.1 SDD).

В режиме редактирования индексной карточки в верхней части страница (область 1 на рис. К.9) располагаются интерфейсные элементы, предназначенные для выбора авторов книги из списка всех авторов, хранящихся в БД (Available authors). Ниже предусмотрены поля (область 2 на рис. К.9) для редактирования названия УММ, области знания, аннотации, комментария, названия издательства, года, числа страниц, для загрузки файла УММ и файла обложки, а также для редактирования номера УДК. В нижней части страницы (область 3 на рис. К.9) располагаются командные кнопки «Delete» (переход к варианту использования «Удаление УММ»), «Save and add another» (переход к варианту использования «Добавить новый УММ»). Назначение командных кнопок «Save and continue editing» и «Save» вытекает из их названия и не требует дополнительного разъяснения.

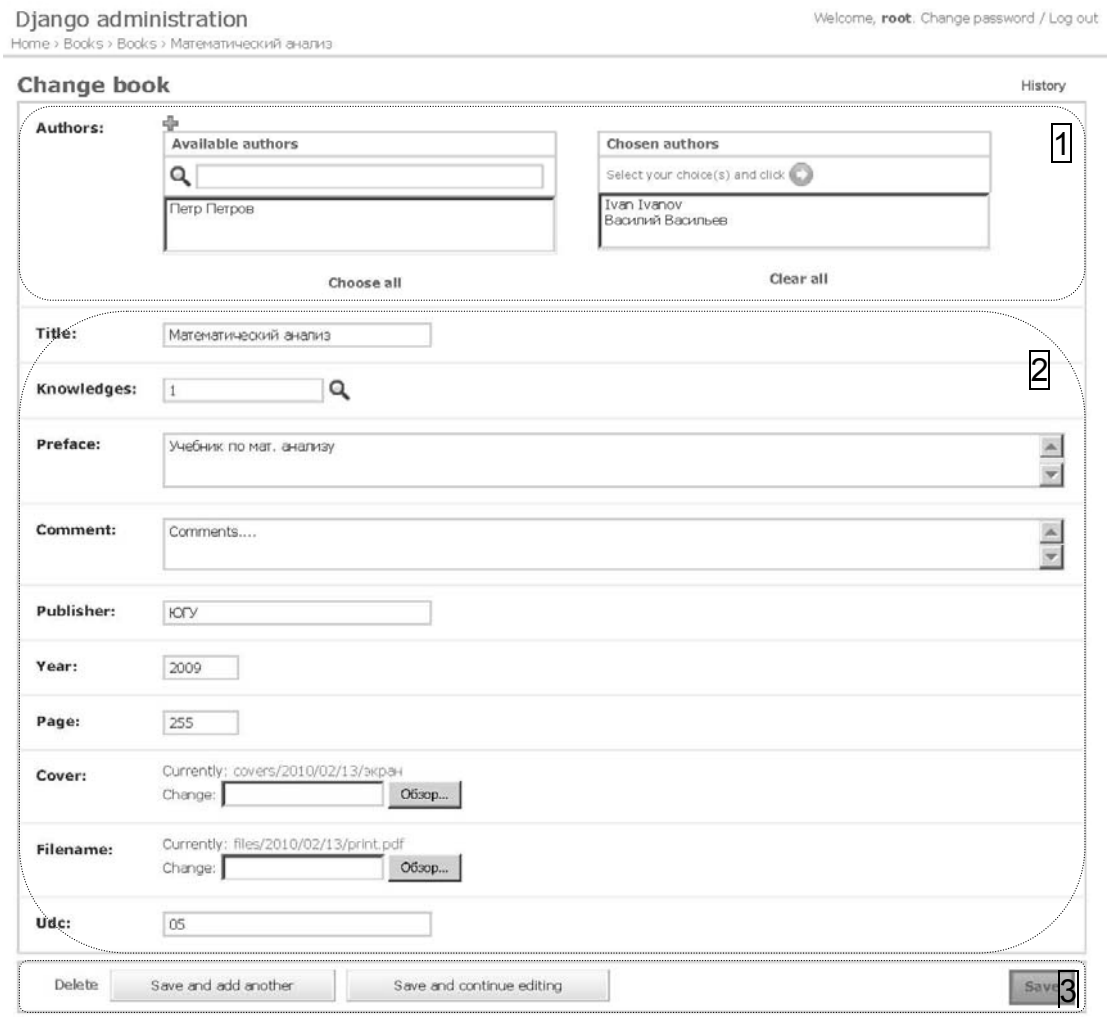

Рис. К.9. Итоговый эскиз ГПИ варианта использования «Редактировать индексную карточку»

3.1.1.6. ГПИ варианта использования П.6 «Добавить новый УММ»

…

3.1.2. Аппаратные интерфейсы

Нет.

3.1.3. Программные интерфейсы

Нет.

3.1.4. Коммуникационные интерфейсы

Нет.

3.2. Описание детальных требований

[*В этом разделе UML диаграммы, полученные на этапе формулирования С-требований (п. 2.2.), уточняются и дополняются атрибутами, предназначенными для описания особенностей системы*]

3.2.1. Диаграмма вариантов использования

[*Для каждого варианта использования в разделе D-требований приводится спецификация*]

На рис. К.10 приведена итоговая (уточненная) версия диаграммы вариантов использования ИС УММ, на которой показан актер *Читатель*, имеющий базовый набор функций, который наследуется *Администратором*, имеющим, в свою очередь, три дополнительных функции: редактировать индексную карточку, добавить новый УММ и удалить УММ. Как видно из диаграммы, пользователю предоставляется возможность скачивания УММ из индексной карточки. Для этого вариант использования П.4 «Скачать УММ» был подключен к варианту использования П.3 «Просмотреть индексную карточку УММ» с помощью отношения <<include>>. В свою очередь, для просмотра индексной карточки необходимо выбрать требуемый УММ посредством выполнения поиска, следовательно, между вариантом использования П.3 и П.2 «Поиск УММ» также необходимо использовать отношение <<include>>. Аналогичные рассуждения использовались при анализе вариантов использования П.5 «Редактировать индексную карточку» и П.7 «Удалить УММ».

3.2.1.1. Вариант использования П.1 «Добавить файл УММ для обработки»

Основной исполнитель.

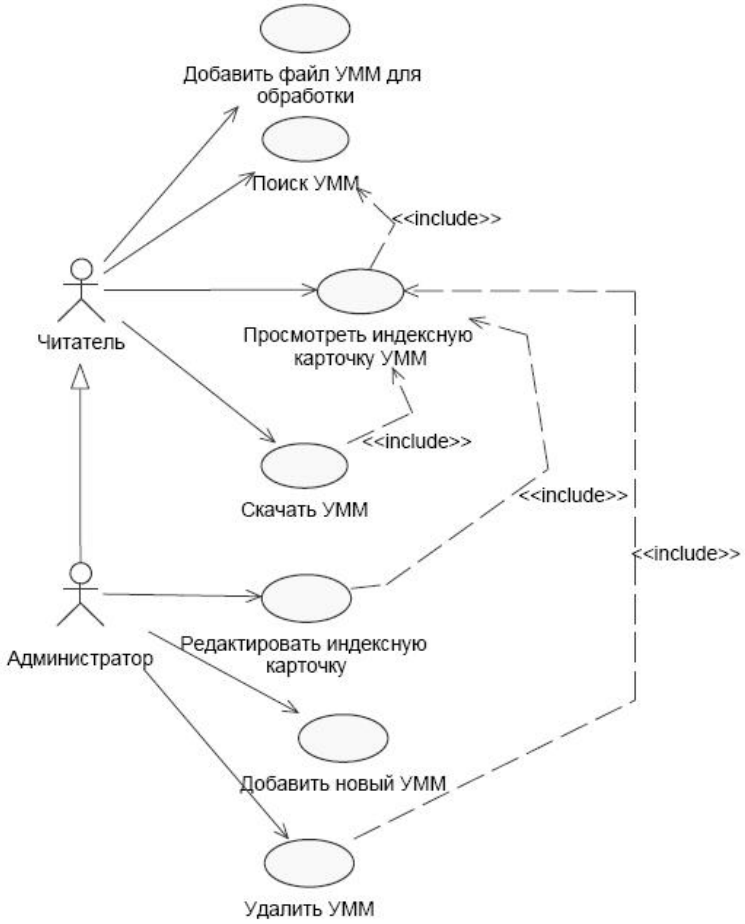

Рис. К.10. Уточненная диаграмма вариантов использования ИС УММ

Администратор.

Заинтересованные лица и их требования: *Администратор* – размещает файлы УММ *Читателя* в соответствующие каталоги системы и добавляет описание УММ в БД системы. Должен убедиться в том, что добавляемый УММ отсутствует в системе. Нуждается в информации о дате загрузки файла УММ на сервер и логине *Читателя*, загрузившего файл.

Читатель – помещает файл УММ, отсутствующий в БД системы, в специальный каталог сервера. Нуждается в информации о добавлении или отказе в добавлении УММ в БД системы.

Предусловия. *Администратор* и *Читатель* идентифицированы и аутентифицированы в системе. *Читатель* с помощью варианта использования «Поиск УММ» убедился в отсутствие добавляемого им УММ в системе.

Постусловия (результаты). Файл УММ *Читателя* помещен *Администратором* в соответствующий каталог сервера. В БД создана запись, представляющая описание УММ, в объеме, предусмотренном системой. Запись проиндексирована для последующего поиска. Файл УММ доступен *Читателям* для скачивания. Из таблицы загруженных *Читателями* УММ удалена запись об УММ, добавленном в БД системы. *Читателю* направлено письмо с уведомлением о результате обработки загруженного им файла.

Основной (успешный) поток.

1. *Читатель* с помощью меню системы вызывает стандартный диалог открытия файла, посредством которого указывает файл УММ, загружаемый на сервер.
2. Система, используя путь к загружаемому файлу, копирует его с диска *Читателя* на сервер в каталог «Upload». При этом на экране появляется информация, свидетельствующая о ходе процесса копирования файла.
3. Для полученного файла система создает новую запись в таблице БД, содержащую следующие данные: имя файла, дата загрузки, логин *Читателя*, загрузившего файл.

…

Альтернативные потоки.

1.а. Размер файла превышает установленное ограничение.

1. Система уведомляет об ошибке и предотвращает загрузку указанного *Читателем* файла.

1.б. Длина имени файла превышает установленное ограничение.

1. Система уведомляет об ошибке и предлагает переименовать файл.

2.а. Нарушение сетевого соединения.

1. Система уведомляет об ошибке сетевого соединения и предлагает пользователю начать загрузку файла сначала.

…

Специальные требования. Сразу после помещения *Администратором* файла УММ *Читателя* в соответствующий каталог сервера система должна автоматически продублировать этот файл в резервном хранилище.

Список технологий и типов данных.

Частота использования. Время от времени. Существует высокая вероятность загрузки файлов одновременно от нескольких *Читателей*.

Открытые вопросы. По какому протоколу «ftp» или «http» будет лучше загружать файлы? Как изменяется скорость загрузки файла при увеличении пользователей, одновременно копирующих файлы на сервер? …

3.2.1.2. Вариант использования П.2 «Поиск УММ»

…

3.2.2. Диаграммы последовательностей

3.2.2.1. Диаграмма последовательностей варианта использования П.1 «Добавить файл УММ для обработки»

Уточненная версия диаграммы последовательностей варианта использования «Добавить файл УММ для обработки» приведена на рис. К.11. Для добавления УММ, отсутствующего в системе, *Читатель* с помощью команды меню указывает путь к файлу УММ. При этом генерируется сообщение *createNewFileUMM()*, которое создает новый объект *Файл УММ*. Объект *Файл УММ* позволяет получить путь (*getPath()*), загрузить файл на сервер (*uploadToServer()*), и вызвать операцию вставки записи в спецификацию загрузок УММ. После того, как будет скопирован указанный пользователем файл в специальный каталог на сервер, в спецификацию загрузок УММ добавляется информация об имени файла, дате загрузки и логине *Читателя*, загрузившего файл (*insertIntoUploadSpec()*). *Администратор* с помощью меню системы сможет просмотреть список всех загруженных файлов (*GetList()*), открыть тот или иной файл для просмотра и добавить файл, снабдив его описанием (название, изд-во, год и т.д. с помощью вызова *insertIntoSpec()*) в соответствующий каталог системы (предварительно убедившись, что такой УММ в системе отсутствует с помощью *searchUMM()*). Если содержание файла не соответствует тематике системы, *Администратор* удаляет файл из каталога сервера вместе с записью об этом файле из *Спецификации загрузок* (*deleteFromUploadSpec(id)*). По окончании обработки файла УММ *Администратор* посылает *Читателю* почтовое уведомлением (*sendNotice()*).

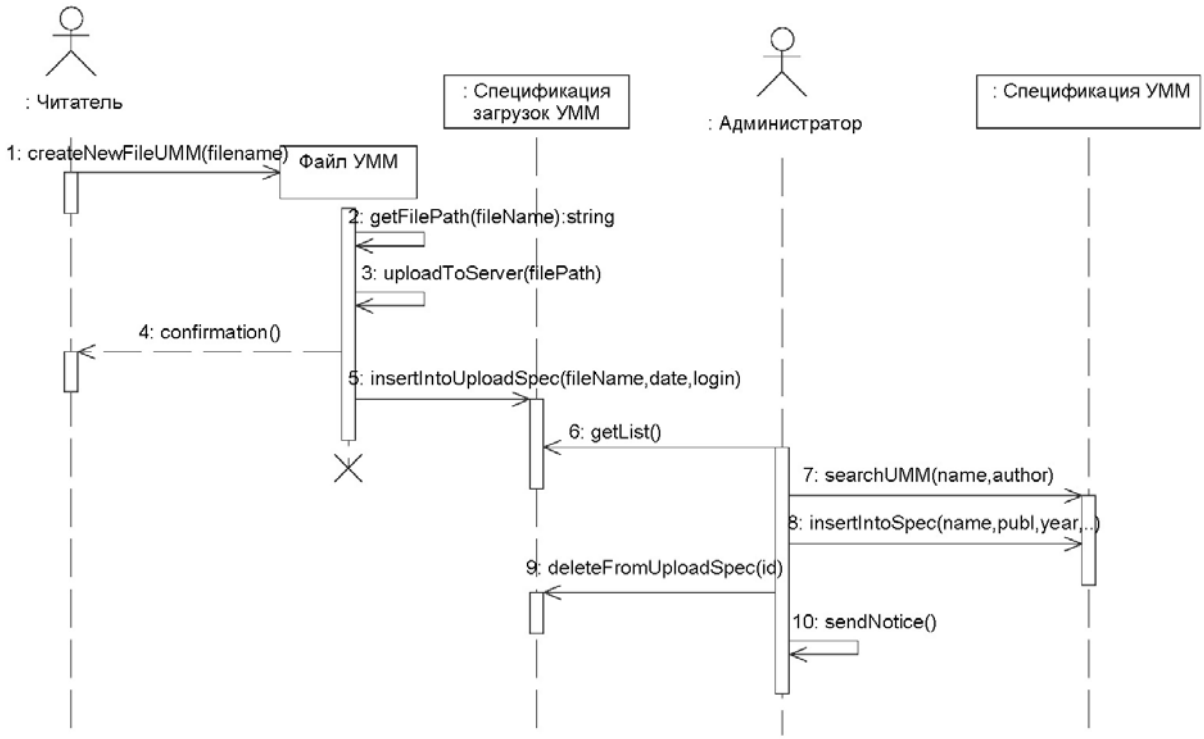

Рис. К.11. Диаграмма последовательностей для варианта использования П.1 «Добавить файл УММ для обработки»

3.2.2.2. Диаграмма последовательностей варианта использования П.2 «Поиск УММ»

…

3.2.3. Диаграммы классов

[*Необходимо описать назначение и взаимосвязь основных классов (классов анализа), полученных по результатам анализа предметной области. Уточненные диаграммы классов описываются и анализируются в главе «Эскизный проект», результаты анализа помещаются в SDD*]

3.2.3.1. Диаграмма классов варианта использования П.1 «Добавить файл УММ для обработки»

Уточненная диаграмма классов представлена на рис. К.12. Символы множественности, размещенные у полюсов ассоциаций (рис. К.12), позволяют подчеркнуть ограничения, накладываемые системой на процессы взаимодействия между экземплярами классов. В частности, следует отметить, что *Читатель* в определенный момент времени может загружать только один файл УММ на сервер, однако, следует иметь в виду, что несколько *Читателей* одновременно должны иметь возможность загружать файлы УММ на сервер.

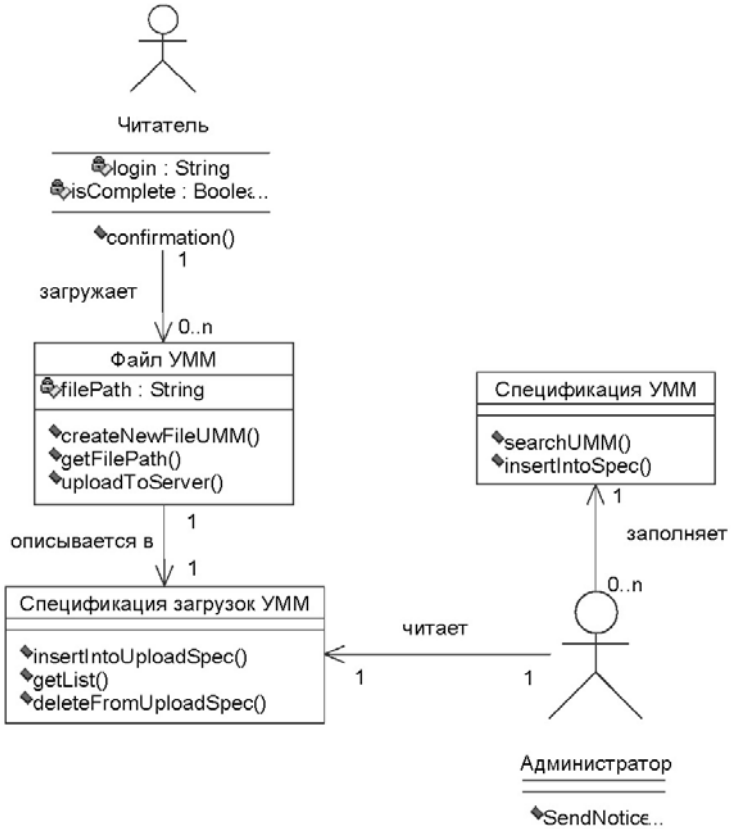

Рис. К.12. Диаграмма классов варианта использования П.1 «Добавить файл УММ для обработки»

3.2.3.2. Диаграмма классов варианта использования П.2 «Поиск УММ»

…

3.2.4. Диаграмма деятельностей

3.2.4.1. Диаграмма деятельности варианта использования П.1 «Добавить файл УММ для обработки»

На диаграмме рис. К.13 для нетривиальных деятельностей показаны внутренние действия и условия их выполнения: в процессе (on do) или по выходу (on exit). Структура уточненной диаграммы деятельностей не претерпела изменений по сравнению с эскизным вариантом (рис. К.8), поэтому в дополнительных комментариях не нуждается.

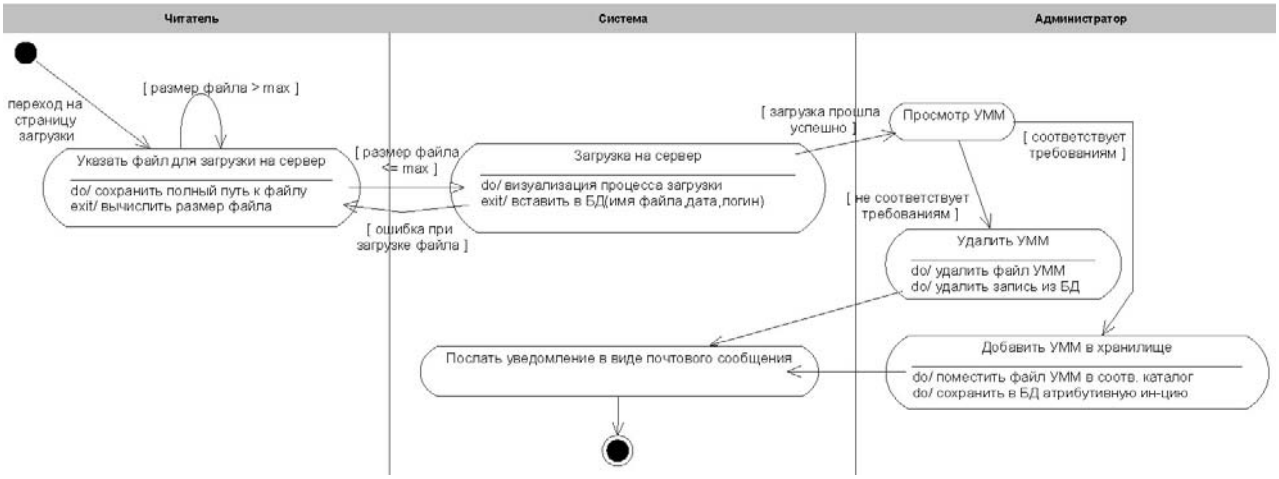

Рис. К.13. Диаграмма деятельности варианта использования П.1 «Добавить файл УММ для обработки»

3.2.4.2. Диаграмма деятельности варианта использования П.2 «Поиск УММ»

…

3.3. Требования к производительности

[*Требования к производительности определяют временные ограничения, которые должны быть выполнены в программе. Заказчики и разработчики обсуждают ограничения по времени вычислений, использованию оперативной памяти, объему запоминающих устройств и т. д., если это не было документировано в другом разделе SRS.*]

ИС должна выдавать результаты поиска менее чем за пять секунд.

3.4. Ограничения проектирования

3.5. Атрибуты программной системы

3.5.1. Надежность

[*Требования надежности определяют надежность в измеряемых величинах. Требования такого типа предполагают вероятность неидеальной работы программы и ограничивают область ее несовершенства*.]

3.5.2. Доступность

3.5.3. Защита

[*Перечисляются требования к обеспечению безопасности системы с учетом возможности искажения данных посредством несанкционированного доступа, сбоя системного или прикладного ПО*.]

Возможность добавления и модификации информации в БД ИСС должна предоставляться авторизованным пользователям, обладающим соответствующими правами.

3.5.4. Поддержка

[*Задаются требования, касающиеся адаптации системы в случае изменения набора или состава бизнес правил. Например, необходимость в изменении алгоритма начисления пени, суммы страховки или поддержка нескольких схем оформления графического интерфейса*]

3.6. Дополнительные требования

Нет.

4. Дополнительная информация

Нет.

4.1. Оглавление и индекс

4.2. Приложения

[*Приложения могут содержать следующее: пример форматов ввода-вывода; результаты опросов пользователей; дополнительную информацию, которая может помочь читателям SRS; специальные инструкции кодирования и способы защиты, экспорта, начальной загрузки и т.д.*]

ПРИЛОЖЕНИЕ Л

(справочное)

IEEE 1016-1987 ПРОЕКТНАЯ ДОКУМЕНТАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (SOFTWARE DESIGN DOCUMENT – SDD)

Утверждаю

_____Дата_____

01.03.09 А. Мелихов: первоначальный эскиз
04.04.09 А. Мелихов: составлена общая схема
06.06.09 А. Мелихов: выявление дефектов
07.08.09 А. Мелихов: добавление компонентов, предложенных А. Мелиховым
08.09.09 А. Мелихов: дополнительная разбивка на компоненты согласно вариантам использования и модели переходов состояний

1. Введение

1.1. Цель

В данном документе представлены результаты проектирования информационно-справочной системы (ИСС).

1.2. Описание проекта

Этот проект представляет собой прототип ИС, создаваемый с учебной целью, на котором демонстрируются приемы разработки архитектуры, детального проектирования и составления документации.

1.3. Определения, сокращения и термины

ИС – информационная система.
CMF – (Content Management Framework) – каркасная система управления сайтом (фреймворк).
ORM – (Object Relational Mapping) – объектно-реляционное отображение.

2. Ссылки

Software Engineering: an Objected-Oriented Perspective. E. Braude, Wiley, 2000.
UML: The Unified Modeling Language User Guide. G. Booch, J. Rumbaugh, I. Jacobson, Addison-Wesley, 1998.
Стандарт IEEE 1016-1987 устанавливает основные направления разработки SDD.

3. Описание декомпозиции

Для описания архитектуры ИС могут использоваться любые средства, позволяющие изобразить функциональную или программно-функциональную структурные схемы (см. определения ГОСТ 24.103-84) архитектуры ИС. Однако, в SDD результаты проектирования архитектуры необходимо по возможности документировать с помощью диаграмм пакетов UML.

3.1. Модульная декомпозиция

[*Описываются архитектурные слои: (i) представления (presentation); (ii) предметной области (domain - домен); (iii) источника данных (data source). Распределение классов по указанным слоям должно быть выполнено таким образом, чтобы полностью исключить зависимость бизнес-логики и источника данных от слоя представления.*]

С целью выбора платформы для реализации проектируемой системы на страницах ПЗ были описаны три каркасных системы управления сайтом (Content Management Framework - CMF): Ruby on Rails, Django, Grails. Как показал анализ наилучшими характеристиками для выполнения данного проекта обладает CMF Django.

CMF Django построена с использованием распространенной архитектуры (паттерна проектирования) Модель-Представление-Контроллер (Model-View-Controller -- MVC). MVC упрощает разработку приложений посредством его деления на несколько слоев, каждый из которых имеет соответствующие роли и возможности. Слой модели отвечает за поддержание состояния приложения. Он инкапсулирует данные приложения и бизнес логику. Слой представления обеспечивает реализацию пользовательского интерфейса приложения. Слой управления интерпретирует обращение пользователей и ответ пользователям посредством передачи данных между слоем модели и слоем представления.

Представление MVC-паттерна при реализации web-приложения показано на рис.Л.1. В контексте web-приложения, слой представления соответствует web-страницам, которые составляют пользовательский интерфейс web-приложения. Слой управления идентифицирует HTTP-запросы и взаимодействует со слоем модели. Слой модели взаимодействует с базой данных и выполняет необходимые запросы бизнес логики к манипулируемым данным.

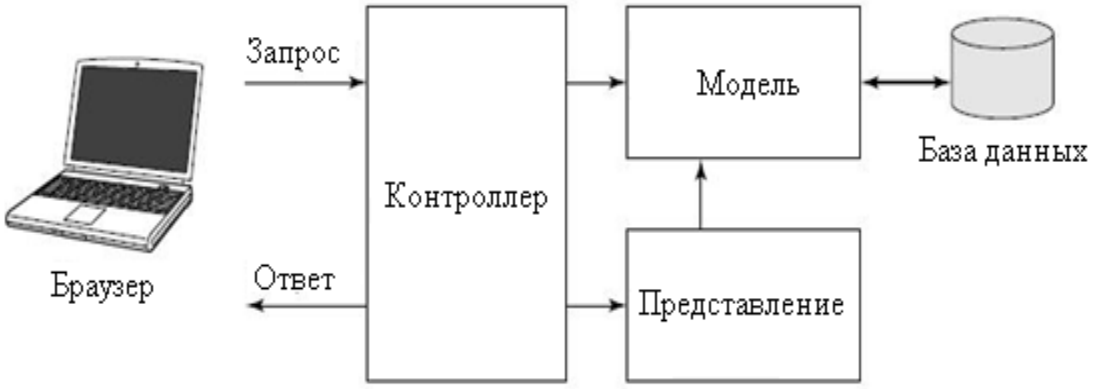

Рис. Л.1. MVC-паттерн web-приложения

В Django реализован механизм автоматического преобразования между объектами модели и таблицами БД – Object-Relational Mapping (ORM).

С программной точки зрения кроме файла представления (views.py), файла модели (model.py) и файла настройки проекта (settings.py) в состав типичного проекта Django входят: (i) шаблоны (templates*. html), которые позволяют определить управляющую логику и встроенные переменные в HTML странице; (ii) файл отображения – URL mapping (url.py) - используются для преобразования запросов от браузера в вызовы функций представления; (iii) файл административного интерфейса (admin.py) определяет структуру и состав интерфейса администратора сайта.

Содержание и структура каждого из перечисленных выше модулей будут описаны в раздел 6 настоящего документа.

3.2. Декомпозиция на параллельные процессы

[*При наличии в системе параллельных процессов выполняется проектирование потоков управления с целью выявления в системе параллельных процессов, порядка их создания и уничтожения, определения характера взаимодействия между собой и т.д.*]

3.3. Декомпозиция данных

[*В этом разделе приводится описание структур данных: исходные данные; результаты функционирования; данные, использующиеся для взаимодействия между подсистемами*]

4. Описание зависимостей

[*В этом разделе дается описание зависимостей между различными вариантами декомпозиции, например, зависимости между пакетами или между страницами, составляющими web-сайт.*]

Использование CMF Django позволяет фактически полностью исключить необходимость анализа зависимостей и проектирование системных интерфейсов. Внимание разработчика сосредотачивается исключительно на вопросах обеспечения необходимого функционала. В этой связи для данного проекта описание зависимостей и системных интерфейсов не приводится.

4.1. Межмодульные зависимости (объектная модель)

4.2. Межпроцессные зависимости

4.3. Зависимости внутри данных

4.4. Зависимости между состояниями

4.5. Зависимости между уровнями

5. Описание интерфейса

[*В этом разделе описываются интерфейсы объектной модели*]

5.1. Межмодульные интерфейсы

[*Проблема организации межмодульных интерфейсов играет принципиальную роль для удовлетворения принципа «слабое внешнее связывание, сильное внутреннее сцепление». Особое значение вопросы проектирования интерфейсов приобретают при проектировании распределенных приложений.*

В настоящем разделе необходимо описать результаты проектирования межмодульных интерфейсов с использованием структурных GoF-паттернов (адаптер, заместитель, фасад) и результаты

проектирования межслойных интерфейсов (для распределенного приложения) с использованием удаленного вызова процедур или асинхронного обмена сообщениями.]

MVC-структура фреймворка Django строго отделяет архитектурные слои приложения друг от друга, а правила построения каждого слоя в отдельности исключают необходимость дополнительной проработки вопросов проектирования межмодульных интерфейсов.

5.2. Интерфейс процессов

Взаимодействие параллельных процессов в проектируемой системе не рассматривается, поскольку вопросы возможных взаимоблокировок при одновременной записи информации в БД несколькими пользователями будут решаться средствами СУБД.

6. Детальное проектирование каркаса ИС

6.1. Детальное проектирование модулей

[*В этом разделе приводятся все необходимые нетривиальные свойства модулей, описанных в разделе 3.1 SDD для каркаса проекта. При описании слоев проводится детальное проектирование классов, состоящее в уточнении атрибутов классов (указывается тип, значение по умолчанию, видимость, инварианты и т.д.), а также операций классов (краткое описание операции, видимость, область действия, предусловия, постусловия*)]

В настоящем разделе приводится описание содержания и структуры каждого из перечисленных в разделе 3.1. модулей.

6.1.1. Слой модели ИС

В настоящем проекте слой модели ИС располагается в файле (models.py). Согласно идеологии CMF Django файл модели предназначен для реализации объектно-реляционного преобразования (ORM). В этой связи структура models.py в общем случае представляет собой объявление классов, относящихся к модели предметной области. На рис. Л.2 приведена диаграмма классов, составляющая модель предметной области проектируемой системы.

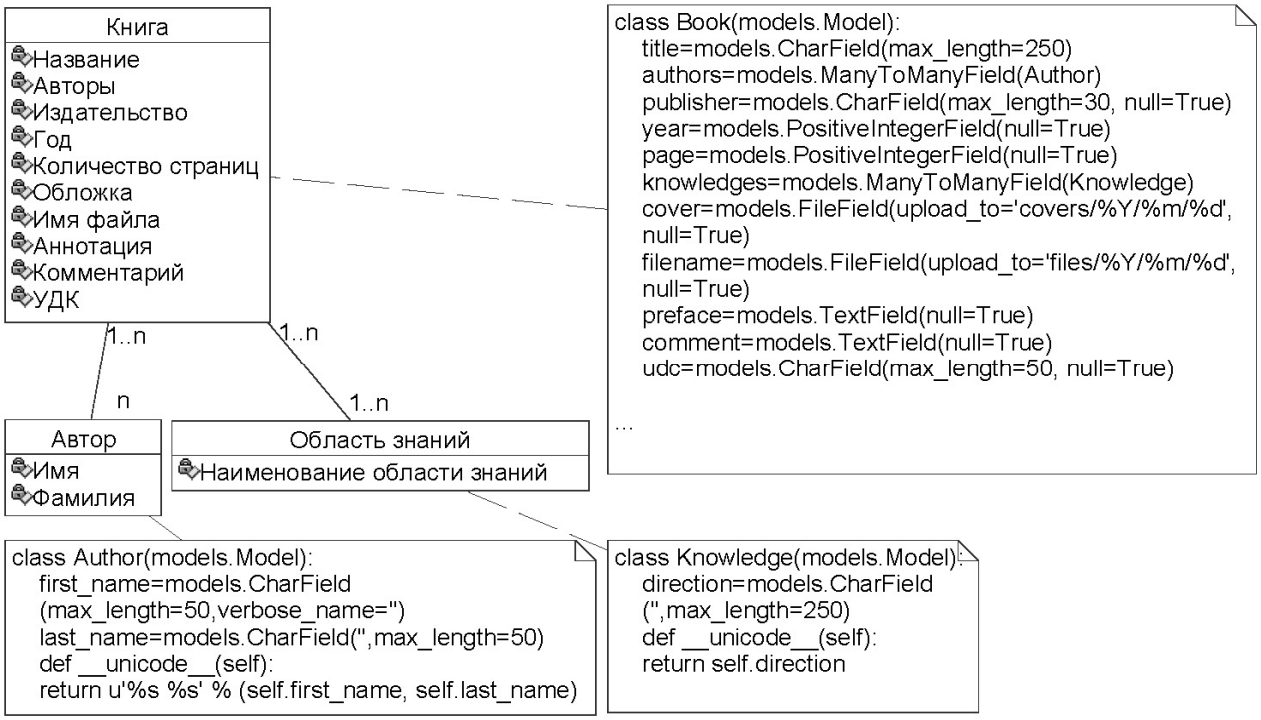

Рис. Л.2. Диаграмма классов модели предметной области

На диаграмме классов (рис. Л.2) в виде комментариев приведены листинги определения классов (с указанием типа атрибутов) слоя модели ИС в соответствии с синтаксисом языка Python. Следует отметить, что для создания зависимости между классами типа «многие ко многим» в Django предусмотрен специальный метод – ManyToManyField, позволяющий сопоставить атрибуты, требующие использование указанного типа связи. Состав и назначение атрибутов классов, приведенных на рисунке Л.2., определены в задании на КП и в дополнительных комментариях не нуждаются.

6.1.2. Слой представления ИС

…

6.2. Детальное проектирование данных

[*Описывается структура БД с использованием ERD или UML; перечисляются все запросы, которые необходимо предусмотреть в системе для ее полноценного функционирования, а также приводят синтаксис и описание типовых SQL-запросов, хранимых процедур, представлений и триггеров, позволяющих, с одной стороны, осуществлять мониторинг за состоянием БД, с другой стороны, решить задачу построения сложно-структурированных запросов к БД, тем самым повысив надежность и расширяемость системы, а также упростив программирование клиентской части приложения.*]

Механизм объектно-реляционного преобразования (ORM), реализованный в CMF Django, позволяет полностью абстрагироваться от взаимодействия с БД, работая исключительно и только с объектной моделью ИС. При этом разработчик web-приложения не привязан к какому-либо диалекту SQL или СУБД.

ПРИЛОЖЕНИЕ М

(справочное)

ОБРАЗЕЦ ОФОРМЛЕНИЯ ТИТУЛЬНОГО ЛИСТА ОТЧЕТА ПО ИНСПЕКТИРОВАНИЮ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

Министерство образования и науки Российской Федерации
Государственное образовательное учреждение высшего профессионального образования
Югорский государственный университет
Институт прикладной математики, информатики и управления
Кафедра «Автоматизированные системы обработки информации и управления»

ОТЧЕТ ИНСПЕКТИРОВАНИЯ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ ПО КУРСОВОМУ ПРОЕКТУ

Дисциплина: «Проектирование автоматизированных систем обработки информации и управления»
(специальность 230102 – Автоматизированные системы обработки информации и управления)

на тему: _____

Группа _____

Студент-исполнитель: ФИО студента _____
(подпись)

Студент-инспектор: ФИО студента _____
(подпись)

Ханты-Мансийск 200_ г.

Составитель:
Мелихов Артем Юрьевич, к.т.н.

ПРОЕКТИРОВАНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

Методические указания по курсовому проектированию
для студентов очной формы обучения направлений
230100 – Информатика и вычислительная техника
010400 – Прикладная математика и информатика
010200 – Математика и компьютерные науки

Издается в авторской редакции с предоставленного автором оригинал-макета

Подписано в печать 22.04.2010 Формат 60x84/16.
Усл. п. л. 5,5. Гарнитура Times New Roman.
Тираж 50 экз. Заказ № 116.

Информационно-издательский центр ЮГУ,
628012, Ханты-Мансийский автономный округ,
г. Ханты-Мансийск, ул. Чехова, 16