Государственное бюджетное образовательное учреждение
высшего образования Московской области
«Университет «Дубна»
Филиал «Протвино»
Кафедра «Информационные технологии»

И. О. Ковцова, А. В. Мандрик, В. И. Ухов

Подготовка курсовых работ по дисциплине «Технология разработки программного обеспечения»

Электронное методическое пособие

Рекомендовано
кафедрой информационных технологий
филиала «Протвино» государственного университета «Дубна»
в качестве методического пособия для студентов,
обучающихся по направлению
«Информатика и вычислительная техника»

Протвино
2017

ББК 32.973.23 я 73
К56

Рецензент:
доктор физико-математических наук,
ведущий специалист, начальник группы ООО «СИСТЕЛ»
С.Р. Слабоспицкий

Ковцова, И.О.
К56 Подготовка курсовых работ по дисциплине «Технология разработки программного обеспечения»: электронное методическое пособие / И.О. Ковцова, А.В. Мандрик, В.И. Ухов. — Протвино: 2017. — 74с.

Предназначено для студентов очного и заочного отделений направления «Информатика и вычислительная техника».

В пособии рассматриваются правила выполнения, определяются требования к содержанию, структуре и оформлению курсовых работ по дисциплине «Технология разработки программного обеспечения», выполняемых на кафедре информационных технологий.

Требования настоящего методического пособия обязательны для всех преподавателей кафедры, ведущих руководство курсовыми работами по данной дисциплине и для всех студентов, выполняющих курсовые работы.

ББК 32.973.23 я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
1 ОБЩИЕ ТРЕБОВАНИЯ К КУРСОВОЙ РАБОТЕ 5
1.1 Правила выбора темы для курсовой работы 5
1.2 Результаты и защита курсовой работы. 6
1.3 Критерии оценки курсовой работы 7
2 ОРГАНИЗАЦИЯ УЧЕБНОГО ПРОЕКТА 9
2.1 Инструментальные средства 9
2.2 Основы командной разработки программного обеспечения 9
2.3 Проект. Обсуждение целей и миссии 19
2.4 Выбор модели жизненного цикла программного обеспечения 21
2.5 Анализ требований и определение спецификаций программного обеспечения 28
2.6 Планирование проекта 33
2.7 Проектирование программного обеспечения 39
2.8 Разработка программного обеспечения 50
2.9 Тестирование программного обеспечения 53
2.10 Подготовка документации к проекту 57
3 ПРОГРАММНЫЕ ТЕХНОЛОГИИ УПРАВЛЕНИЯ ПРОЕКТАМИ 61
3.1 *GanttProject* 61
3.2 *On-line* сервисы управления проектами 61
3.3 *Redmine* 61
3.4 *Atlassian JIRA* 62
4 РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ 62
4.1 Структура курсовой работы 63
4.2 Оформление курсовой работы 66
4.3 Рекомендации по разработке презентации 70
Библиографический список 73
Приложения 74
Приложение А ОБРАЗЕЦ ТИТУЛЬНОГО ЛИСТА КУРСОВОЙ РАБОТЫ 74

Введение

Целью преподавания дисциплины «Технология разработки программного обеспечения» является освоение студентами теоретических основ, принципов и современных методов разработки программного обеспечения в соответствии с российскими и международными стандартами и практикой, а также — получение компетенций и практических навыков по проектированию, документированию, тестированию, внедрению, сопровождению и модернизации программных продуктов и систем. Работая над утвержденным курсовым заданием, студенты осваивают методы организации работ в коллективе, способы планирования, контроля и проведения аудита проекта.

Главная задача курса состоит в развитии у студентов понимания процесса разработки программного обеспечения, то есть студент должен иметь представление о существующих технологиях разработки ПО, о моделях процесса разработки, о стандартах и методологиях построения моделей предметных областей с целью их использования в процессе анализа требований к программному обеспечению. Основной акцент при изучении дисциплины делается на практической части, позволяющей выработать у студентов, как навыки самостоятельной работы, так и навыки работы в команде.

Итоговой работой при изучении курса «Технология разработки программного обеспечения» является курсовая работа — проект, который предполагает разработку небольшой завершенной программной системы или продукта. Работа над проектом ведётся сформированной группой студентов. Данная дисциплина изучает-ся студентами на последнем, выпускном курсе и предусматривает знакомство с основными технологиями создания программного обеспечения, изучавшимися ранее. В первую очередь студенты должны владеть навыками:

— программирования и отладки;

— проектирования и создания БД;

— создания распределенных программных систем;

— разработки дружественных человеко-машинных интерфей-сов.

Реализуемый проект обязательно должен опираться на современные информационные технологии.

Выполнение курсовой работы дает возможность:

— научить студентов применять знания на практике в различ-ных областях, выявить у них исследовательские навыки, а также способность адаптироваться к новым ситуациям;

— привить студентам навык анализировать и синтезировать полученную информацию;

— осваивать навыки формирования команды и работы в ней;

— научиться ставить задачи, формулировать их и грамотно производить декомпозицию работ;

— развивать умение грамотно пользоваться языком предметной области;

— выявлять общие формы, закономерности и инструментальные средства данной дисциплины;

— развить творческие способности студентов а также — помочь найти своё место в проекте;

— формирует целеустремлённость, ответственность, инициативность и умение как принимать решения, так и решать возникающие проблемы;

Курсовая работа является завершающим этапом изучения дисциплины “Технология разработки программного обеспечения”, в ходе которого осуществляется обучение применению полученных знаний и умений при решении комплексных за-дач, связанных с профессиональной деятельностью будущих специалистов, хорошо подготовленных для производственно-технологической, организационно-управленческой, проектной и научно-исследовательской деятельности.

Методические рекомендации предназначены для студентов очного и заочного отделений направления «Информатика и вычислительная техника».

1 Общие требования к курсовой работе

1.1 Правила выбора темы для курсовой работы

Темы курсовых работ разрабатываются преподавателем и утверждаются заведующим кафедрой в течение первого месяца семестра, в котором выполняются курсовые работы. Затем темы выдаются студентам.

Каждая группа в течение семестра должна полностью выпол-нить один проект. В рамках данной дисциплины студенты могут предлагать темы курсовых работ самостоятельно.

Проект, выбранный группой, должен удовлетворять ряду требований. Во-первых, объем работ по проекту должен быть таким, чтобы выполняющая его группа могла выполнить его в течение семестра. Вторым требованием к проекту, являются технологические ограничения. Предлагаемый студентами проект должен опираться на современные программные технологии. Для успеш-

ной реализации проекта выбранные технологии должны быть известны студентам и доступны для использования на практических занятиях. У студентов должна быть возможность получить расширенные консультации по их использованию от преподавателей кафедры. Допускается привлечение в качестве консультантов специалистов сторонних организаций, занимающихся данной предметной сферой, соответствующей теме курсовой работы. Третьим требованием к проекту является его инновационность и креативность. Для того, чтобы студенты получили прочные навыки рабо-ты в группе, выполняемая работа должна их увлекать и достав-лять радость, выполняемый проект должен быть интересен для всей команды. Еще одним желательным требованием к проекту является его практическая польза. Как и для большинства учеб-ных задач, данное требование не является обязательным.

Приступая к выполнению курсовой работы, студенты выполняют декомпозицию работ по проекту. Каждый участник получает индивидуальное задание в рамках проекта, которое согласуется с преподавателем.

Руководителем курсовой работы является преподаватель, ведущий семинарские занятия. Преподаватель оказывает помощь по вопросам содержания и последовательности выполнения работы, консультации по проекту, а также осуществляет контроль своевре-менности и качества выполняемых работ.

1.2 Результаты и защита курсовой работы

В результате выполнения курсовой работы студенты должны предоставить преподавателю законченный программный проект, документацию к нему и набор артефактов, созданных в процес-се его разработки. Также предоставляется презентация проекта в электронном виде. Отчет о курсовой работе в виде файла формата *MS Word* или аналогичном из *Open Office* предоставляется не позднее, чем за две недели до установленного срока защиты курсовой работы. Отчёт оформляется в соответствии с правилами, описанными в п. 4. На основании этих материалов аттестуется проект в целом. Также определяется личный вклад и индивидуальные оценки каждого участника проекта. Если данный программный продукт был внедрён или находился на опытной эксплуатации в сторонней организации, то желательно приложить отзыв о программном продукте (это может быть заключение или рекомендательное письмо заказчика, сторонних экспертов).

Не допускаются к защите и возвращаются для повторного написания курсовые работы, полностью или в значительной сте-

пени, выполненные не самостоятельно (путём механического копирования материала из источников информации, более 50%), а также работы, характеризующиеся низким уровнем грамотности и небрежным оформлением.

Защита курсовой работы проводится до начала зачётной сессии. Защита может проходить открыто с приглашением преподавателей и студентов. Студенты демонстрирует подготовленную презентацию проекта. В данной презентации должны быть отражены миссия и цель проекта, постановка задачи, изложены основные требования, предъявляемые к разработанному программному продукту, описаны существующие и выбранные методы и сред-ства его реализации, их достоинства и недостатки, а также должен быть проиллюстрирован процесс разработки ПО и полученные результаты. Обязательно должен быть продемонстрирован сам проект, программный продукт. На выступление и представление проекта отводится время в пределах не более семи минут. В конце сообщения необходимо сделать краткое заключение по достигнутым результатам работы.

При изложении материала студент должен продемонстрировать:

— умение кратко, четко и технически грамотно излагать содержание проекта;

— умение обосновать с инженерной точки зрения выбранный вариант решения, технологии, алгоритма и т.д.;

— владение теоретическим материалом по предмету курсовой работы;

— хорошее владение математическим аппаратом и четкое ори-ентирование в расчетах.

После сообщения студент отвечает на вопросы руководителя и присутствующих, касающиеся темы курсовой работы.

1.3 Критерии оценки курсовой работы

Оценка за курсовую работу учитывает:

1. Актуальность, новизну и практическую ценность работы.
2. Функциональность и степень завершённости программного продукта.
3. Полноту раскрытия темы, основательную проработку исследуемой предметной области.
4. Удобство и дружественность разработанного программного продукта.
5. Применение современных информационных технологий в процессе создания ПО.

6. Применение программных технологий управления проектом.

7. Наличие артефактов разработки ПО.

8. Соблюдение структуры курсовой работы, правильность оформления, использование современных компьютерных выразительных средств, грамотность, хороший язык и стиль изложения.

9. Качество презентации, умение коротко и ясно рассказать о своей работе.

10. Понимание и знания, проявленные при ответах на вопросы во время защиты.

Рекомендуемые критерии оценки:

— «отлично» — оцениваются работы, в которых содержатся элементы научного и продуктивного творчества. Участники проекта подготовили полезный с практической точки зрения продукт, разработали четкие рекомендации по его примене-нию, результаты практической работы полезны и заслуживают внимания. Проведён анализ существующих решений, и в ходе системного анализа и инновационного подхода, был получен новый более качественный программный продукт с расширенной функциональностью. Если данный продукт внедрён или находится на стадии тестирования, для последующей эксплуатации. В ходе разработки применялись современные информационные технологии. Оценка «отлично» выставляется студенту, показавшему глубокие знания и навыки, примененные им при самостоятельной работе над своей частью проекта.

— «хорошо» — оцениваются работы, в которых наблюдается незавершенность программного продукта, отсутствие артефактов разработки ПО. Нет анализа существующих подобных решений. Не обозначены преимущества выбранной технологии разработки программного продукта. Не полно и не всесторонне освещаются вопросы предметной области, не был проявлен творческий подход. Оценка «хорошо» выставля-ется студенту, не проявившему самостоятельность при работе над его частью проекта. Основанием для снижения оценки мо-жет так же служить нечеткое изложение целей и результатов проекта, затруднения при ответах на вопросы.

— «удовлетворительно» — оцениваются работы, в которых правильно освещены основные вопросы и темы, но студент не проявил необходимой доли самостоятельности, умения последовательно излагать вопросы и логично рассказывать о проделанной работе, или допустил отдельные ошибочные высказывания.

— «неудовлетворительно» — получает студент, не владею-щий материалом, не раскрывший основные положения избран-ной темы и допустивший грубые ошибки в содержании рабо-ты, при невразумительных ответах на поставленные вопросы. Работа может быть оценена отрицательно, если установлено, что она является плагиатом, а содержание работы не соответ-ствует заявленной теме.

При получении неудовлетворительной оценки работа должна быть переработана с учетом высказанных замечаний и представлена на защиту в сроки, установленные руководителем. Студенты, получившие по курсовой работе неудовлетворительную оценку, к сдаче экзамена по данной дисциплине не допускаются.

2 Организация учебного проекта

2.1 Инструментальные средства

Настоятельно рекомендуется использование в проектах следующих классов инструментальных средств.

1. Электронная почта.
2. Централизованное хранение файлов (возможно, система управления версиями).
3. Представление данных о ходе проекта в виде *web*-ресурса.
4. Инструментальные средства разработки ПО.
5. Редакторы (текстовые, графические, моделей) и другие средства документирования, включая *web*-средства.
6. Компиляторы, отладчики, редакторы связей, менеджеры сборки и конфигурации.
7. Стандартные библиотеки и оболочки разработки ПО.
8. Системы тестирования и статического анализа кода.

Возможно предоставление и других средств, в зависимости от специфики проекта. Поощряется самостоятельное нахождение, изучение и использование студентами специализи-рованных инструментальных средств (в рамках существующего законодатель-ства РФ).

2.2 Основы командной разработки программного обеспечения

Разработка программного обеспечения — это групповой процесс. Большинство программных продуктов рождается в результате коллективных усилий. Люди являются одной из главных составляющих, необходимых для производства программного продукта. При организации работы программистов важно учитывать не только их технические навыки, но и взаимодействие между ними.

Существует две основные модели организации коллектива при разработке ПО:

— *Иерархическая модель* определяет начальников и подчиненных, то есть — это структура с вертикальной формой управления (контроля) элементами, входящими в неё. Фактически это пирамида, каждым уровнем которой управляет более высокий уровень. Можно выделить следующие недостатки иерархической модели:

♦ нехватка информации на различных уровнях;

♦ невозможность учесть все особенности проекта;

♦ отсутствие полноценной связи между всеми участниками проекта, так как вся информация идет в одном направлении — вверх по иерархии, к главному менеджеру;

♦ трудность освоения новых технологий, необходимых при создании кроссплатформенных приложений;

♦ сложность расстановки приоритетов.

— *Модель проектной группы*. В проектной группе предусматривается распределение обязанностей руководителя между членами коллектива. При этом за проект отвечает не один человек, а все члены группы — каждый за свой участок.

Можно выделить следующие недостатки модели группы:

♦ разрозненная связь с внешними источниками информации;

♦ несогласованное представление о разных сторонах проекта;

♦ несогласованность личных планов членов группы; ♦ отсутствие опыта, снижающее эффективность коллективной работы.

При разработке программного обеспечения выполняемые задачи распределяются по семи ролевым группам и шести ролям. Участники команды, выполняющие конкретную роль, должны рассматривать проект с точки зрения своей специализации и обладать необходимой квалификацией.

Каждая ролевая группа выполняет свои задачи:

«Архитектура»

♦ формулирует спецификацию решения и разрабатывает его архитектуру;

♦ определяет структуру развертывания (внедрения) решения.

«Разработка»

♦ определяет детали физического дизайна;

♦ оценивает необходимые время и ресурсы на реализацию каждого элемента дизайна;

♦ разрабатывает или контролирует разработку элементов; ♦ подготавливает продукт к внедрению; ♦ консультирует команду по технологическим вопросам.

Рисунок 1 — Роли в модели проектной группы

«Тестирование»

- ♦ обеспечивает обнаружение всех дефектов;
- ♦ разрабатывает стратегию и планы тестирования;
- ♦ осуществляет тестирование.

«Управление выпуском»

- ♦ представляет интересы отделов поставки и обслуживания продукта;
- ♦ организует снабжение проектной группы;
- ♦ организует внедрение продукта;
- ♦ вырабатывает компромиссы в управляемости и удобстве со-провождения продукта;
- ♦ организует сопровождение и инфраструктуру поставки.

«Удовлетворение потребителя»

- ♦ представляет интересы потребителя в команде;
- ♦ организует работу с требованиями пользователя;
- ♦ определяет компромиссы, относящиеся к удобству использо-вания и потребительским качествам продукта;

- ♦ определяет требования к системе помощи и её содержание;
- ♦ разрабатывает учебные материалы и осуществляет обучение пользователей.

«Управление продуктом»

- ♦ выступает в роли представителя заказчика;
- ♦ организует работу с требованиями заказчика;
- ♦ формирует ожидания заказчика;
- ♦ формирует общее видение и рамки проекта;
- ♦ определяет компромиссы между параметрами "возможности продукта / время / ресурсы";
- ♦ организует маркетинг;
- ♦ разрабатывает, поддерживает и исполняет план коммуникаций.

Влияние на успех группы такого важного фактора, как состав команды, выявили М. Белбин и его коллеги. Было проведено исследование нескольких сотен небольших групп в процессе их дея-тельности. Учёные определили, что поведение членов групп соот-ветствует одной из девяти выделенных ими в ходе исследования ролей (Таблица 1).

Таблица 1 — Типология командных ролей М. Белбина

Виды командных ролей	**Необходимые личностные качества и вклад в деятельность команды**	**Допустимые недостатки**
Мыслитель (генератор идей)	Творческая направленность, богатое воображение, неординарность мышления. Стремление к новаторству. Источник оригинальных идей для команды.	Недостаточность опыта межличностного общения. Психологическая неустойчивость. Может долго задерживаться на рассмотрении «интересных идей».
Исполнитель	Претворяет идеи в практические действия. Превращает решения в легко выполнимые задания. Вносит упорядоченность в деятельность команды.	Недостаточная гибкость. Неприязнь к фантастическим идеям. Неприязнь к частым изменениям планов.

Продолжение таблицы 1

Виды командных ролей	Необходимые личностные качества и вклад в деятельность команды	Допустимые недостатки
Доводчик	Усердие и добросовестность. Следит за тем, чтобы задания выполнялись полностью. Отслеживает своевременность выполнения заданий.	Чрезмерная обеспокоенность состоянием дел. Склонность к внутренним переживаниям. Нежелание перепоручать свои обязанности. Неприятие несерьезного отношения к его обязанностям со стороны других.
Оценщик (эксперт)	Исповедует беспристрастный критический анализ ситуации. Стратегический подход и проницательность в оценках. Точность суждений, стремление рассматривать все возможные варианты решения.	Недооценка факторов стимулирования и воодушевления. Недостаточность вдохновения и творческого воображения. Способность сбивать других, подавляя их инициативу.
Исследователь ресурсов	Владение искусством проведения переговоров, разнообразие контактов. Талант импровизатора, изучает благоприятные возможности. Энтузиазм, коммуникабельность.	Теряет интерес по мере угасания энтузиазма. Перескакивает от одной задачи к другой. Нуждается в повышенном внешнем давлении.
Формирователь	Постоянная ориентированность на решение поставленной задачи; стимулирует работу всей команды. Способствует реализации принятых решений; побуждает сотрудников работать интенсивнее. Энергичность, стремление к превосходству и работе с полной отдачей сил.	Легко переходит к состоянию раздражительности. Импульсивность и нетерпеливость. Нетерпимость к нечетким формулировкам и нерешительности в поведении. Результат - любой ценой.

Продолжение таблицы 1

Виды командных ролей	Необходимые личностные качества и вклад в деятельность команды	Допустимые недостатки
Коллективист	Способствует гармонизации отношений в команде и устранению разногласий. Внимательно выслушивает собеседника; опирается на мнения других. Чуткость, отсутствие чрезмерной самоуверенности.	Нерешительность в кризисных ситуациях. Стремление избегать обострения ситуаций. Может воспрепятствовать совершению действий в решающий момент.
Председатель (координатор)	Четко формулирует цели; хорошо выполняет функции ведущего во время дискуссий. Способствует эффективному принятию решений. Имеет хорошие коммуникативные навыки; социальный лидер.	Может производить впечатление человека, склонного к манипуляциям. Склонность к переложению своих обязанностей на других. Может приписывать себе заслуги всей команды.
Специалист	Обладает редко встречающимися навыками и знаниями. Целеустремленность и способность концентрировать усилия. Инициативность и способность всецело отдаваться работе.	Полезен только в узкой профессиональной сфере. Зачастую слабые коммуникативные навыки. Иногда, образно выражаясь, «не видит леса за деревьями».

В равной степени важными для эффективной командной работы их можно считать лишь при условии, что они применяются в надлежащее время и наилучшим образом. Например, в тот период, когда команда только приступает к рассмотрению пробле-мы или к разработке проекта, обычно требуются инновационные идеи (то есть нужен «мыслитель»). Затем возникает необходи-мость оценки того, как эти идеи могут быть претворены в практи-чески выполнимые задачи (то есть нужен «исполнитель»). Успех достигается при условии, что в команде есть хороший координа-тор («председатель»), задача которого — обеспечить наибольшую отдачу от членов команды в нужное время. Движущую силу и стимул команда обретает в лице энергичного «формирователя». Ког-да возникает необходимость в проведении сложных переговоров

с другими группами, большое значение приобретают качества, которыми обладает «исследователь ресурсов». Чтобы сдержи-вать чрезмерные проявления энтузиазма, отвлекающие от главной цели, в составе команды должен быть «оценщик» (то есть. «экс-перт»). Всевозможные трения и недоразумения между членами команды устраняются усилиями «коллективиста». Благодаря на-личию «специалиста» команда имеет редко встречающиеся навы-ки и знания, в которых периодически возникает необходимость. Роль «доводчика» состоит в том, чтобы не упустить из виду даже мельчайшие детали реализации всех намеченных планов.

Однако для успешной разработки программного обеспечения недостаточно просто распределить среди участников проектной команды роли и обязанности. Необходимо, чтобы все участники команды понимали и соблюдали следующие базовые принципы коллективной работы:

— Единое понимание целей проекта. Данное условие является важнейшим фактором успеха как при разработке ПО, так и при любой совместной деятельности. На начальных этапах работы каждый участник команды имеет собственное мнение и представление о продукте. Необходимо обсудить частные мнения и по итогам обсуждения выработать концепцию проекта — единый до-кумент, который содержит:

♦ описание возможностей продукта (что продукт позволяет сделать);

♦ конкретизацию продукта (описание функциональных возможностей данной версии);

♦ описание путей реализации проекта.

— Равенство, свободный обмен информацией, общая ответственность и четкие ролевые функции. В группе разработчиков важна каждая роль. Только такой подход позволяет наладить сво-бодный обмен информацией между проектными группами и от-дельными разработчиками и сформировать общую ответствен-ность за качество продукта проекта.

— Ориентация на продукт и понимание целей заказчика, на-целенность на результат.

— Постоянное коллективное обучение. Обучение людей является одной из важнейших составляющих успеха проекта и всей компании. Постоянное коллективное обучение позволяет членам проектной группы извлекать пользу из отрицательного опыта сделанных ошибок, равно как и воспроизводить успехи, используя проверенные методы работы других людей. На разных этапах реализации проекта применяются следующие виды обучения:

♦ обучение на опыте других проектов;
♦ изучение методологии;
♦ изучение технологий.

— Эффективное взаимодействие с внешними группами. Проектная группа должна эффективно взаимодействовать с внешни-ми группами — с заказчиком, пользователями, другими разработчиками. За эффективность внешних взаимодействий отвечают менеджер проекта, менеджер продукта, специалист по внедрению и логистик. В их обязанности входят как внутренние, так и внешние контакты.

Можно выделить следующие **этапы развития команды:**

1. Первый этап — **формирование** (или шумиха) — характеризуется избытком энтузиазма. Люди должны преодолеть внутрен-ние противоречия, переболеть конфликтами прежде, чем сформи-руется действительно спаянный коллектив. На этом этапе многое зависит от лидера: он должен четко поставить цели членам коман-ды, верно определить их роли в проекте.

2. Второй этап — **бурление** (или неразбериха) — самый сложный и опасный период. Мотивация новизны уже исчезла, а силь-ные и глубокие стимулы у команды еще не появились. Неизбеж-ные сложности или неудачи порождают конфликты, «поиск вино-вных». На этом этапе важно обеспечить открытую коммуникацию в команде — конфликты не следует прятать или разрубать. Споры необходимо разруливать — спокойно, терпеливо и тщательно.

3. Третий этап — **становление**. В команде растет доверие, люди начинают замечать в коллегах не только проблемные, но и сильные стороны. На смену битве амбиций приходит продуктивное сотрудничество. Четче становится разделение труда, исчезает дублирование функций. Лидер перестает находиться в состоянии постоянного аврала, работа по построению команды на этом эта-пе — уже не тушение пожара, а скрупулезный труд по отработке общих норм и правил.

4. Четвертый этап — **отдача**. Наконец-то можно получить дивиденды за потраченные усилия. Команда работает эффективно, высок командный дух, люди хорошо знают друг друга и умеют использовать сильные стороны коллег. Высок уровень доверия. Это лучший период для раскрытия индивидуальных талантов. Люди хотят и могут совершенствоваться, они более всего заинтересова-ны в профессиональном росте. Растет значение нематериальной мотивации сотрудников, а оценивать и поощрять материально лучше команду в целом. Если четвертый этап достигнут, значит, команда создавалась не зря.

Любая коллективная разработка программного обеспечения сталкивается с одними и теми же проблемами:

— групповая работа над кодом, документами;

— учет проблем, ошибок, требований;

— документирование, накопление и циркуляция (поиск, транс-ляция, агрегация) знаний компании;

— организация правильного тестирования.

Работа средств коллективной разработки основана на выполнении двух базовых функций:

— Ведение общей для всех разработчиков базы данных проекта, содержащей историю изменений в каждом файле проекта.

— Автоматизация следующих операций:

♦ доступа к общей базе данных;

♦ обработки конфликтующих версий файла;

♦ именования различных версий файла;

♦ ввода и сохранения комментариев к изменениям.

Методические рекомендации

Формирование рабочих коллективов и выбор программного проекта — вещи взаимосвязанные. В данном курсе даётся возможность студентам сначала «собрать команду», а потом выбрать проект. Группы, работающие над проектом, должны состоять из 3-5 человек. Группы менее 3-х человек, не позволят студентам по-лучить навыки коллективной работы над проектом. Группы более 5-ти человек требуют более сложной координации и допустимы для уже сложившихся творческих коллективов.

При формировании групп следует учитывать психологическую совместимость входящих в них студентов. Задача преподавателя — помочь студентам получить навыки конструктивной работы в коллективе. При формировании и утверждении групп следует избегать как существующих, так и потенциальных конфликтов внутри одной группы.

В сильные сформированные группы преподаватель может по собственному усмотрению добавить студента, не нашедшего группу самостоятельно. Студенты, не сумевшие сформировать группы, разделяются на группы преподавателем, либо вводятся в уже существующие группы.

Сформированные группы утверждаются преподавателем. Если преподаватель считает, что с точки зрения освоения практи-ческих навыков, предусмотренных программой ТРПО (Техноло-гия разработки программного обеспечения), данная группа мало-

эффективна, то он вправе расформировать или переформировать такую группу. Использовать данное право следует только в исключительных случаях, предварительно проконсультировавшись с лектором.

Студенты, на основе своих личных предпочтений и желаний, организуют рабочие команды. Следует выбирать тех людей, с ко-торыми приятно и интересно работать. После выбора темы про-екта рекомендуется распределить роли между участниками, дого-вориться о способах взаимодействия, а также определить количе-ство часов в неделю, выделяемых на работу над проектом каждым участником. Данный момент является очень важным, поскольку одной из распространенных причин провала проекта является не-выполнение рабочих обязательств одним или несколькими члена-ми группы**.**

Следует определить схему и частоту проведения собраний. Встреча коллектива, совместно работающего над проектом, долж-на проходить не реже одного раза в неделю. Собрание является точкой интеграции полученных результатов проекта и определе-ния дальнейших действий проекта. На собрании обсуждаются и вырабатываются различные решения, касающиеся проекта. Например, распределение задач, определение приоритетов, назначение ответственных, изменение плана и т.п.

Рекомендуется следующая схема проведения собрания:

1. Руководитель проекта назначает секретаря и оглашает заранее подготовленную повестку собрания и краткий отчет о теку-щем состоянии проекта. (~5 минут).

2. Участники собрания вносят замечания. В результате форми-руется окончательная повестка собрания. (~5 минут).

3. Собрание идет согласно выработанной повестке. Заслушиваются отчеты разработчиков о проделанной работе, включая отчеты об исправлении замечаний, выполняется анализ текущей документации и других артефактов. Обсуждаются и определяются дальнейшие направления и задачи. (~35–70 минут).

4. Подведение итогов. Распределение задач, назначение ответ-ственных и т.п. (— 10 минут).

Рекомендуется вести журнал собраний, в котором указывается дата встречи, обсуждаемые вопросы, принятые решения и т.п.

В случае недостаточности еженедельных собраний для командной работы разработчиков, команда может использовать дополнительное время. Например, это полезно на начальных этапах проекта в случае активной работы с заказчиком.

Разработчики сочетают командную работу с индивидуальной. В процессе разработки каждому из разработчиков должны поручаться конкретные задачи для решения. Зоны ответственности, роли разработчиков и другие правила, регулирующие индивидуальную работу и ее сочетание с командной, фиксируются в плане проекта. Текущее распределение задач отражается в протоколах собраний и индивидуальных журналах разработчиков.

2.3 Проект. Обсуждение целей и миссии Что же такое проект? Все мы постоянно осуществляем проекты в своей повседневной жизни. Вот простые примеры: ремонт в квартире, стройка на даче, проведение исследований, написание книги. Само слово проект имеет греческие корни и означа-ет — брошенный вперёд, выдающийся вперёд. В настоящее время проект определяется следующим образом:

Проект — это уникальный процесс, состоящий из совокупности скоординированных и управляемых видов деятельности с начальной и конечной датами, предпринятый для достижения цели, соответствующей конкретным требованиям, включающий ограничения по срокам, стоимости и ресурсам.

Для осуществления проекта его выполнение разбивают на различные этапы. Например, при разработке программного обеспечения часто выделяются такие этапы как осознание потребности в информационной системе, формулирование требований, проектирование системы, кодирование, тестирование, эксплуатацион-ная поддержка. Наиболее общим является разбиение проекта на четыре крупных этапа: формулирование проекта, планирование, осуществление и завершение.

Одной из основных задач этапа формулирования проекта является определение миссии и целей проекта.

Миссия — это генеральная цель проекта, четко выраженная причина его существования. Миссия отражает наивысшее предназначение проекта, а также выражает его общественную значимость. Она детализирует статус проекта, обеспечивает ориентиры для определения целей следующих уровней, а также стратегий на различных организационных уровнях. Миссия — это главная задача проекта, с точки зрения его будущих основных услуг или изделий, его важнейших рынков и преимущественных технологий.

Сущность миссии состоит в том, что она:

— отражает общие ценности и взгляды членов коллектива;

— связана с культурой организации;

— определяет направленность процесса принятия решений и работы;

— формулируется таким образом, чтобы можно было оценить степень ее реализованости.

Миссия отвечает на вопросы: кто?, что?, для кого? и как (каким образом)?

Цели проекта намного более конкретны, чем миссия. Они определяют результаты, которые необходимо получить для того, чтобы была выполнена общая миссия. Цели проекта — это работа, которую необходимо сделать для производства продукта (услуги), обладающего требуемыми свойствами. Цель, которую предстоит достичь, должна быть конкретной, измеримой, достижимой, ре-альной, ограниченной во времени.

Цель становится *задачей*, если указан срок ее достижения и заданы количественные характеристики желаемого результата. Задача должна являться логическим следствием поставленной проблемы, и быть направленной на её решение для достижения поставленной цели (причинно-следственная связь). Задачи представляют собой конкретные промежуточные измеряемые результаты в ходе реализации проекта.

Методические рекомендации

Предварительно сформировав проектные группы, студенты определяют тему курсовой работы. В качестве отправной точки можно использовать приведённый в приложении список рекомендованных тем. В случаях, когда группа не пришла к единому мнению по поводу темы, можно изменить состав группы, либо обратиться за помощью к преподавателю. Преподаватель может назначить тему директивным способом, или предложить тему, максимально отвечающую областям компетенции членов группы. Тема проекта обязательно утверждается преподавателем в тече-ние первых двух недель.

После прохождения процедуры утверждения темы курсовой работы требуется провести следующие проектные мероприятия:

— разработать формулировку миссии;

— сформулировать главные цели проекта;

— зафиксировать границы проекта (что должно быть сделано, а чего делать не надо);

— произвести декомпозицию работ;

— оценить сроки деятельности, необходимые ресурсы.

— завести рабочую тетрадь проекта;

2.4 Выбор модели жизненного цикла программного обеспечения

Жизненный цикл программного обеспечения (ЖЦ ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Модель жизненного цикла программного обеспечения— струк-тура, определяющая последовательность выполнения и взаимос-вязи процессов, действий и задач на протяжении жизненного цик-ла. Модель жизненного цикла зависит от специфики, масштаба и сложности проекта и специфики условий, в которых система соз-даётся и функционирует.

Исторически в ходе развития теории проектирования программного обеспечения и по мере его усложнения утвердились основные модели ЖЦ. Каждая такая модель представляет процесс создания ПО в каком-то своем «разрезе», используя только определенную часть всей информации о процессе. Перечисленные ниже модели разработки представляют обобщенные модели, основанные на архитектурном подходе. В этом случае можно уви-деть всю структуру процесса создания ПО, абстрагируясь от част-ных деталей отдельных составляющих его этапов.

Эти обобщенные модели не содержат точного описания всех стадий процесса создания ПО. Напротив, они являются полезными абстракциями, помогающими «приложить» различные подходы и технологии к процессу разработки. Кроме того очевидно, что процесс создания больших систем не является единым, а состоит из множества различных процессов, ведущих к созданию отдель-ных частей большой системы.

Первой во времени появления и самой распространённой явилась ***каскадная модель*** *(водопад)*. Основные базовые виды деятельности, выполняемые в процессе создания ПО (такие, как разработка спецификации, проектирование и производство , атте-стация и модернизация ПО), представляются как отдельные этапы этого процесса.

Последовательность фаз водопадной модели: анализ требований, проектирование, реализация, интеграция.

Каскадная модель характеризуется следующими основными особенностями:

— последовательным выполнением этапов проекта в строгом фиксированном порядке;

— началом последующего этапа только после завершения предыдущего;

— отсутствием обратных связей между этапами;

— получением результата только в конце разработки.

Каскадная модель хорошо функционирует при её применении в циклах разработки программного продукта, в которых используется неизменяемое определение продукта и вполне понятные технические методики. Модели данного типа на протяжении всего времени их существования используются при выполнении больших проектов, в которых задействовано несколько больших команд разработчиков.

В случае ***спиральной модели*** последовательность фаз*:* анализ требований, проектирование, реализация, тестирование — вы-полняется больше одного раза. Для этого может быть несколь-ко причин. Основная причина обычно связана с необходимостью предупреждения рисков. Другой причиной может быть необходи-мость предоставить заказчику частичную версию проекта для по-лучения отзывов и пожеланий. Если разрабатываемая программа достаточно сложна, необходимо выполнять промежуточные инте-грации, не откладывая эту фазу на самый конец, как это предпи-сывает водопадная модель. Общая же идея спирального процесса заключается в том, чтобы на каждой итерации строить очередную версию программы, используя в качестве основы её предыдущую версию. В этом случае процесс приобретает спиралевидный ха-рактер.

Основной особенностью ***итерационной модели*** ЖЦ ПО, или так называемой поэтапной модели с промежуточным контролем является наличие обратных связей между этапами, вследствие этого появляется возможность проведения контроля и корректировок на каждой стадии разработки.

В ***эволюционной модели*** разработки ПО последовательно перемежаются этапы формирования требований, разработки ПО и его аттестации. «Быстрая» частичная реализация системы создаётся перед этапом определения требований или на его протяжении. Конечные пользователи системы используют ускоренный прототип, а затем путём обратной связи сообщают о своём достижении команде, работающей над проектом, для дальнейшего уточнения требований к системе. Процесс уточнения продолжается до тех пор, пока пользователь не получит то, что ему требуется. После завершения процесса определения требований путём разработки ускоренных прототипов, получают детальный проект системы, а ускоренный прототип регулируется при использовании кода или внешних утилит, в результате чего получают конечный рабочий продукт.

Инкрементальная модель жизненного цикла**.** Инкрементная разработка представляет собой процесс частичной реализации всей системы и медленного наращивания функциональных возможностей или эффективности. Этот подход позволяет уменьшить затраты, понесенные до момента достижения уровня исходной производительности. С помощью этой модели ускоряется процесс создания функционирующей системы. Этому способствует при-меняемый принцип компоновки из стандартных блоков, благода-ря которому обеспечивается контроль над процессом разработки изменяющихся требований.

Инкрементная модель описывает процесс, при выполнении которого первостепенное внимание уделяется системным требованиям, а затем их реализации в группах разработчиков. Как правило, со временем инкременты уменьшаются и реализуют каж-дый раз меньшее количество требований. Каждая последующая версия системы добавляет к предыдущей определённые функцио-нальные возможности до тех пор, пока не будут реализованы все запланированные возможности. В этом случае можно уменьшить затраты, контролировать влияние изменяющихся требований и ускорить создание функциональной системы благодаря использованию метода компоновки из стандартных блоков.

Предполагается, что на ранних этапах жизненного цикла выполняется конструирование системы в целом. На этих этапах определяются относящиеся к ним инкременты и функции. Каждый инкремент затем проходит через остальные фазы жизненного цикла: кодирование, тестирование и поставку.

Модель формальной разработки систем. Основана на разработке формальной математической спецификации программной системы и преобразовании этой спецификации посредством специальных математических методов в исполняемые программы. Проверка соответствия спецификации и системных компонентов также выполняется математическими методами.

Модель разработки ПО на основе ранее созданных компонентов. Предполагает, что отдельные составные части программной системы уже существуют, то есть созданы ранее. В этом слу-чае технологический процесс создания ПО основное внимание уделяет интеграции отдельных компонентов в общее целое, а не их созданию. Использование готовых системных компонентов практикуется повсеместно. Данный метод получил распространение, поскольку сборка систем из готовых или ранее использованных компонентов значительно ускоряет разработку ПО.

Выбор приемлемой модели жизненного цикла разработки ПО может осуществляться в ходе следующего процесса.

1. анализ отличительных категорий проекта, помещенных в таблицах ниже;

2. ответьте на вопросы, приведённые для каждой категории, обведя кружочком слова «да» или «нет» (таблица 2, таблица 3, таблица 4, таблица 5);

3. Расположите по степени важности категории или вопросы, относящиеся к каждой категории, относительно выбранного проекта;

4. воспользуйтесь упорядоченными категориями для разрешения противоречий, возникающих при сравнении моделей, если общие полученные показатели сходны или одинаковы.

Таблица 2 — Выбор модели жизненного цикла на основе характеристик требований

Требования	Каскадная	Прототипирование	Спиральная	Инкрементная
Являются ли требования легко определимыми и хорошо известными	Да	Нет	Нет	нет
Могут ли требования заранее определяться в цикле	Да	Нет	Нет	да
Часто ли будут изменяться требования в цикле	Нет	Да	Да	нет
Нужно ли демонстрировать требования с целью определения	Нет	Да	Да	нет
Требуется ли для демонстрации возможностей проверка концепции	Нет	Да	Да	нет
Будут ли требования отражать сложность системы	Нет	Да	Да	Да
Обладает ли требование функциональными свойствами на раннем этапе	Нет	Да	Да	Да

Таблица 3 — Выбор модели жизненного цикла на основе характеристик участников команды разработчиков

Команда разработчиков проекта	Каскадная	Прототипирование	Спиральная	Инкрементная
Являются ли проблемы предметной области проекта новыми для большинства разработчиков	Нет	Да	Да	Нет
Является ли технология предметной области проекта новой для большинства разработчиков	Да	Нет	Да	Да
Являются ли инструменты, используемые проектом, новыми для большинства разработчиков	Да	Нет	Да	Нет
Изменяются ли роли участников проекта во время жизненного цикла	Нет	Да	Да	Да
Могут ли разработчики проекта пройти обучение	Нет	Нет	Нет	Да
Является ли структура более значимой для разработчиков чем гибкость	Да	Нет	Нет	Да
Будет ли менеджер проекта строго отслеживать прогресс команды	Да	Нет	Да	Да
Важна ли лёгкость распределения ресурсов	Да	Нет	Нет	Да
Приемлет ли команда равноправные обзоры и инспекции, менеджмент/обзоры заказчиков, а также стадии	Да	Да	Да	Да

Таблица 4 — Выбор модели жизненного цикла на основе характеристик коллектива пользователей

Коллектив	Каскадная	Прототипирование	Спиральная	Инкрементная
Будет ли присутствие пользователей ограничено в жизненном цикле	Да	Нет	Да	Да
Будут ли пользователи знакомы с определением системы	Нет	Да	Да	Да
Будут ли пользователи ознакомлены с проблемами предметной области	Нет	Да	Нет	Да
Будут ли пользователи вовлечены во все фазы жизненного цикла	Нет	Да	Нет	Нет
Будет ли заказчик отслеживать ход выполнения проекта	Нет	Да	Да	Нет

Таблица 5 — Выбор модели жизненного цикла но основе характеристик типа проектов и рисков

Тип проекта и риски	Каскадная	Прототипирование	Спиральная	Инкрементная
Будет ли проект идентифицировать новое направление продукта для организации	Нет	Да	Да	Да
Будет ли проект иметь тип системной интеграции	Нет	Да	Да	Да
Будет ли проект являться расширением существующей системы	Нет	Нет	Нет	Да

Продолжение таблицы 5

Тип проекта и риски	Каскадная	Прототипирование	Спиральная	Инкрементная
Будет ли финансирование проекта стабильным на всём протяжении жизненного цикла	Да	Да	Нет	Нет
Ожидается ли длительная эксплуатация продукта в организации	Да	Нет	Да	Да
Должна ли быть высокая степень надёжности	Нет	Нет	Да	Да
Будет ли система изменяться, возможно, с применением непредвиденных методов, на этапе сопровождения	Нет	Да	Да	Да
Является ли график ограниченным	Нет	Да	Да	Да
Являются ли «прозрачными» интерфейсные модули	Да	Нет	Нет	Да
Доступны ли повторно используемые компоненты	Нет	Да	Да	Нет
Являются ли достаточными ресурсы (время инструменты, персонал)	Нет	Да	Да	Нет

Методические рекомендации

В рамках ранее созданных проектных групп необходимо выбрать наиболее подходящую для вашей группы модель разработки программного обеспечения. Для обеспечения квалифицированно-го выбора участникам группы необходимо изучить предложенные модели, их преимущества и недостатки. При осуществлении выбора стоит ответить на вопросы, в значительной степени обуславливающие выбор. Каковы основные характеристики требований

к проекту (сформулированы ли требования в полном объёме, насколько они изменяемы и т.д.)? Каков состав проектной группы и какова компетенция участников (все ли члены команды владеют предметной областью? Все ли знакомы с предполагаемым инстру-ментарием и т.д.)? Какие ограничения на модель возлагает вы-бранный тип проекта?

Сделанный группой выбор следует обосновать. Также, по результатам реализации проекта, требуется рассмотреть моменты отклонения этапов процесса от эталонных этапов модели.

2.5 Анализ требований и определение спецификаций программного обеспечения Определение корректных требований — это, наверное, самый ответственный этап программного проекта. Очень важно, чтобы формат проекта соответствовал требованиям к ПО, собранным командой разработчиков, в противном случае эти требования не смогут быть поддержаны и представлены в программном продук-те. Можно выделить следующие виды требований к программным продуктам:

— ***Бизнес-требования*** содержат высокоуровневые цели орга-низации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы. В этом доку-менте объясняется, почему организации нужна такая система, т.е. описаны цели, которые организация намерена достичь с ее помо-щью.

— ***Бизнес-правила*** включают корпоративные политики, правительственные постановления, промышленные стандарты и вычислительные алгоритмы. Бизнес-правила не являются требованиями к ПО, потому что они находятся снаружи границ любой системы ПО.

— ***Требования пользователей*** основываются на целях и задачах, которые пользователям позволит решать будущая система. К способам представления этого вида требований относятся вари-анты использования, сценарии, прецеденты, таблицы «событие-отклик» и т.п.

— ***Функциональные требования*** - это перечень функций или сервисов, которые должна выполнять система, причём должно быть указано, как система реагирует на те или иные входные дан-ные, как она ведёт себя в определённых ситуациях и т.п., так же указывается, что система не должна делать, т.е. накладываются ограничения поведение системы. Спецификация функциональ-ных требований включает в себя описание функций, которые не должны быть противоречивыми и взаимоисключающими.

— ***Нефункциональные требования*** связаны с такими интеграционными свойствами системы, как надежность, время ответа или размер системы. Кроме того, нефункциональные требования могут определять ограничения, накладываемые на систему, например на пропускную способность устройств ввода-вывода, или форматы данных, используемых в системном интерфейсе. Нефункциональные требования можно разделить на две большие категории, соответствующие поведенческим и структурным аспек-там приложения. Первая категория содержит атрибуты, имеющие значение при исполнении приложения, то есть в режиме его работы. Вторая категория определяет атрибуты, относящиеся к аспектам проектирования приложения.

Атрибуты качества при исполнении приложения:

— Доступность — требования ко времени непрерывной работы приложения, например, 24x7, минимальное время простоя и т.п.

— Надежность — поведение приложения при наступлении нештатных ситуаций, например, автоматический перезапуск, вос-становление работы, дублирование важных данных, резервирова-ние логики.

— Требования к долговременному хранению результатов работы приложения, например, использование базы данных, требования ко времени продолжительности хранения данных.

— Масштабируемость — требования к горизонтальному или вертикальному масштабированию приложения.

— Требования к удобству использования приложения с точки зрения использования, поддержки.

— Требования к безопасности работы или использования при-ложения, связанные с разграничением доступа, работой с при-ватными данным, снижения подверженности рискам от внешних атак.

— Требования к конфигурируемости работы приложения, вза-имодействия и расположения компонентов.

— Требования к производительности решения, количество одновременно работающих пользователей, обслуживаемых транзакций, времени реакции, продолжительности вычислений, скорости и пропускной способности каналов связи.

— Описание ограничений, накладываемых на объем доступной памяти, процессорного времени, дискового пространства, пропускную способность сети, при которых приложение должно эффективно выполнять возложенные на него задачи.

Атрибуты качества при проектировании приложения:

— Требования к повторному использованию реализации или компонентов приложения, а также реализация приложения с возможностью повторного его использования для различных задач.

— Требования к расширяемости приложения в связи с появле-нием новых функциональных требований.

— Требования к портируемости (переносимости) приложения на различные платформы.

— Требования к взаимодействию между компонентами реше-ния, между внешними компонентами, использование стандарт-ных протоколов и технологий взаимодействия.

— Требования к различным аспектам поддержки приложения, таким как дешевизна и скорость разработки, прозрачность поведения приложения, простота анализа ошибок и проблем в работе

— Требования к разделению приложения на модули.

— Требования к возможности автоматического и ручного тестирования приложения, наличие необходимого инструментария.

— Требования к возможности и простоте локализации приложения, перечень языков, на которые предполагается локализация приложения.

— Особые требования к совместимости между версиями приложений, между различными приложениями и внешними подсистемами.

Также можно привести следующую классификацию нефункциональных требований:

— системные требования и ограничения,

— требования качества,

— внешние системы и интерфейсы.

Процесс формирования и анализа требований проходит че-рез ряд этапов:

1. ***Анализ предметной области***. Аналитики должны изучить предметную область, где будет эксплуатироваться система.

2. ***Сбор требований.*** Это процесс взаимодействия с лицами, формирующими требования. Во время этого процесса продолжается анализ предметной области.

3. ***Классификация требований***. На этом этапе бесформенный набор требований преобразуется в логически связанные группы требований.

4. ***Разрешение противоречий.*** Без сомнения, требования мно-гочисленных лиц, занятых в процессе формирования требований, будут противоречивыми. На этом этапе определяются и разреша-ются противоречия такого рода.

5. ***Назначение приоритетов.*** В любом наборе требований одни из них будут более важны, чем другие. На этом этапе совместно с лицами, формирующими требования, определяются наиболее важные требования.

6. ***Проверка требований.*** На этом этапе определяется их пол-нота, последовательность и непротиворечивость.

Когда группа разработчиков начинает анализировать требования, заказчик обычно всё ещё формирует концепции того, что он хочет и что ему нужно. Это аналогично этапу уточнения требований при взаимодействии архитектора и клиента. Например, кли-ент может захотеть дом с четырьмя спальнями и большой гости-ной. Но при уточнении требований клиент вынужден полагаться на архитектора, который должен помочь клиенту понять, чего последний хочет (например, просторную гостиную, в которой мож-но разместить 10 человек). Нужды заказчика несколько труднее определить, чем его желания. Например, заказчик может захотеть, чтобы музыкальное приложение позволяло компьютерным новичкам гарантированно записывать музыку, но ему также может быть нужна периодическая функция автосохранения для избегания потери работы. Являются ли последняя характеристика требованием, или это часть проектирования? Это зависит от соглашения между разработчиком и заказчиком. Если заказчик, осознав смысл автосохранения, принимает решение, что он действительно хочет иметь такую функцию, тогда автосохранение становится требованием.

Большая часть анализа требований является коммуникационной деятельностью, тщательно организованной для получения наилучших результатов. Выделяют следующие методы выявления требований:

— интервью, опросы, анкетирование;

— мозговой штурм, семинар;

— наблюдение за производственной деятельностью;

— анализ нормативной документации;

— анализ моделей деятельности;

— анализ конкурентных продуктов;

— анализ статистики использования предыдущих версий системы;

Рассмотрим один из способов проведения опроса

заказчика Перед интервью рекомендуется:

— перечислить и расставить приоритеты в списке интервьюи-руемых;

— спланировать интервью с фиксированным временем начала и конца (при этом должны присутствовать, как минимум, два чле-на команды разработчиков);

— приготовиться записывать видео- или аудио-файлы.

Во время интервью:

— сконцентрироваться на слушании:

◦ быть активным: исследовать самому и побуждать собе-седника;

◦ настаивать на понимании желаний и изучении нужд;

◦ обсудить варианты использования, а также потоки данных и диаграмм переходов состояний;

◦ делать подробные заметки;

— спланировать следующую встречу.

После интервью:

— составить черновик требований;

— связаться с заказчиком для получения его замечаний по до-кументу;

Все важные для системы требования документируются в спец-ификации требований к программному обеспечению.

Методические рекомендации

В рамках сформированных проектных групп необходимо про-вести анализ предметной области и наиболее полно и точно сфор-мулировать требования к разрабатываемому программному обе-спечению. Требования целесообразно разделить на два основных подмножества: требования пользователей и требования разработ-чиков. Результатом этапа должна являться спецификация на ПО, составленная в следующем формате:

1. Введение 1.1 Цель (Цель документа в целом, а не цель ПО)
   1.2 Область применения (Какие аспекты программы этот документ должен охватить)
   1.3 Определения, термины и сокращения
   1.4 Ссылки (исходные документы)
   1.5 Обзор
2. Общее описание
   2.1 Перспективы продукта
      2.1.1 Концепции операций
      2.1.2 Концепции пользовательского интерфейса
      2.1.3 Аппаратные интерфейсы
      2.1.4 Программные интерфейсы
      2.1.5 Коммуникационные интерфейсы
      2.1.6 Ограничения по памяти

2.1.7 Операции
2.1.8 Требования по адаптации
2.2 Функции продукта
2.3 Пользовательские характеристики (покажите, какие люди будут типичными пользователями программы)
2.4 Ограничения (все условия, которые могут ограничить возможности разработчика. Могут исходить из разных источников)
2.5 Предположения и зависимости (могут быть сделаны лю-бые допущения)
2.6 Распределение требований
3 Детальные требования
3.1 Требования к внешнему интерфейсу
3.2 Детальные требования
3.3 Требования к производительности
3.4 Ограничения проектирования
3.5 Атрибуты программной среды
3.5.1 Надёжность
3.5.2 Доступность
3.5.3 Защита
3.5.4 Поддержка
3.6 Дополнительные требования
4 Дополнительная информация
4.1 главление и индекс
4.2 Приложение

2.6 Планирование проекта

Планирование — это непрерывный процесс определения наилучшего способа действий для достижения поставленных целей с учетом складывающейся обстановки. От качества планирова-ния во многом зависит результат всего начинания. Большой успех складывается из маленьких достижений, которые особенно важны в начале проекта и позволяют придать ему необходимый импульс и создать нужный настрой у его участников. Частые срывы и изменения плана будут играть отрицательную роль. И наоборот, достижение промежуточных целей позволяет почувствовать уверенность в своих силах и поверить в успех всего проекта.

Планирование определяется как ответ на вопросы:

— Что должно быть сделано?
— Кто сделает?
— Как они это сделают?
— Как долго будут это делать? И т.д.

План проекта — это единый, последовательный и согласованный документ, включающий результаты планирования всех функций управления проектом и являющийся основой для выполнения и контроля проекта. Под планом проекта должна стоять подпись всех членов команды.

Наиболее часто используемый для планирования проекта метод — декомпозиция работ. Декомпозиция работ — процесс разделения, разбивки сложной задачи на несколько более простых. Основная причина неудачи проекта — что-то забытое, декомпози-ция работ помогает избежать этой проблемы, поскольку её основ-ная цель — охватить все задачи.

Календарный план — это план, в котором объёмы работы раз-биты по срокам. Календарный план призван обеспечить осущест-вление проекта в рамках намеченного времени.

Обычный календарный план представляется в виде таблицы со следующей структурой:

Таблица 6 — Календарный план

Наименование работ (тема, работа, задача, задание)	Сроки выполнения начало/конец		Ответственный исполнитель и исполнители, роли	Требуемые ресурсы и сроки их предоставления план/факт	Примечания
	план	факт			
1	2	3	4	5	6

Столбец 1 заполняется в соответствии с разбиением заказанного проекта на составляющие. Обычно глубина разбиения зави-сит от декомпозиции проекта. Распределение времени и контроль над ним — назначение столбцов 2 и 3. В них указываются кален-дарные даты планируемого (столбец 2) и фактического (столбец 3) сроков выполнения работы. Планируемое начало работы — это самая ранняя дата, после которой можно приступать к выполнению; конец — это предельный срок отчета исполнителей перед руководителем. Иногда графа планируемых сроков дополняется критическими и целесообразными сроками начала/конца работы. Столбец 4 «Ответственный исполнитель и исполнители, роли» задает информацию о том, кто работает над данным заданием, и какая квалификация от исполнителей требуется. Распределение технических ресурсов и задание сроков их предоставления — содержание столбца 5. Здесь указывается необходимая для выпол-

нения задания техническая, а в ряде случаев, и программная база. Полезным расширением состава сведений столбца 5 является включение в него информации о зависимости работ внутри про-екта, т.е. перечисление заданий (в том числе, ссылки на другие строки данного календарного плана), без выполнения которых осуществимость планируемых работ нарушается.

Сетевое планирование — набор методов, который предназначен для управления расписанием проекта. Его основной инструмент — сетевой график. Сетевой график — это ориентированный граф, в котором вершинами обозначены работы проекта, а дуга-ми — временные взаимосвязи работ. Сетевой график позволяет:

— выявить перечень работ вашего проекта;

— наглядно представить порядок их следования;

— определить длительности каждой работы и всего проекта;

— определить критические работы проекта и его критический путь;

— определить резервы времени по каждой работе;

— и т.д.

В качестве примера рассмотрим проект «Разработка программного комплекса». Предположим, что проект состоит из работ, характеристики которых приведены в таблице 7.

Таблица 7 — Характеристики проекта

Номер работы	Название работы	Длительность	Зависимость
1	Начало реализации проекта	0	
2	Постановка задачи	10	
3	Разработка интерфейса	5	
4	Разработка модулей обработки данных	7	
5	Разработка структуры базы данных	6	
6	Заполнение базы данных	8	
7	Отладка программного комплекса	5	
8	Тестирование и исправление ошибок	10	
9	Составление программной документации	5	
10	Завершение проекта	0	

Сетевой график позволяет по заданным значениям длительностей работ найти критические работы проекта и его критический путь. ***Критической*** является такая работа, для которой задержка ее начала приведет к задержке срока окончания проекта в целом. Некритические работы имеют некоторый запас времени, и в пределах этого запаса их начало может быть задержано. ***Критический путь*** — это путь от начальной к конечной вершине сетевого графика, проходящий только через критические работы. Суммар-ная длительность работ критического пути определяет минималь-ное время реализации проекта.

Нахождение критического пути сводится к нахождению критических работ и выполняется в два этапа.

1. Вычисление раннего времени начала каждой работы проекта. Эта величина показывает время, раньше которого работа не может быть начата.

2. Вычисление позднего времени начала каждой работы проекта. Эта величина показывает время, позже которого работа не может быть начата без увеличения продолжительности всего про-екта.

Критические работы имеют одинаковое значение раннего и позднего времени начала.

Диаграмма Ганта — это инструмент иллюстрации календарного плана. Представляет собой столбчатые диаграммы, вписанные в календарную шкалу. Длина диаграммы соответствует отрезку времени, отведенному на выполнение той или иной задачи.

Рисунок 2 — Диаграмма Ганта

Впервые эта методика была представлена в 1910 году американским инженером Генри Гантом. Впоследствии диаграмма Ган-та стала главным инструментом, применяемым при календарном

планировании и контроле. В 1990-х годах методика была усовершенствована — для описания зависимостей между задачами были добавлены связи.

Типы связей:

— Финиш-Старт — операция В не может начаться до заверше-ния операции А;

— Финиш-Финиш — операция В должна окончиться не рань-ше операции А;

— Старт-Старт — операция В начинается не раньше операции А;

— Старт-Финиш — операция В не может окончиться пока не начнется операция А.

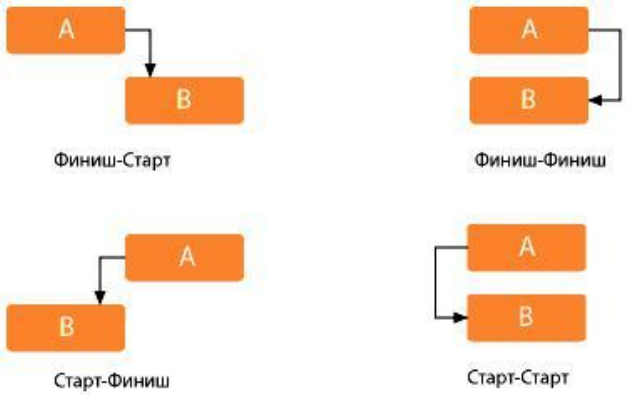

Рисунок 3 — Типы связей

Подобно распределению времени выполнения этапов, менеджер должен рассчитать распределение ресурсов по этапам, в частности — назначить исполнителей на каждый этап. В таблице 8 приведено распределение разработчиков на каждый этап.

Таблица 8 — Исполнители этапов

Этап	Исполнитель
T1	Джейн
T2	Анна
T3	Джейн
T4	Фред
T5	Мэри

Продолжение таблицы 8

Этап	Исполнитель
T6	Анна
T7	Джим
T8	Фред
T9	Джейн
T10	Анна
T11	Фред
T12	Фред

Временная диаграмма распределения работников по этапам

Приведенная таблица может быть использована программны-ми средствами поддержки процесса управления для построения временной диаграммы занятости сотрудников на определенных этапах работ (рисунок 4). Персонал не занят в работе над проек-том все время его реализации. В течение периода незанятости сотрудники могут быть в отпуске, работать над другими проектами, проходить обучение и т.д.

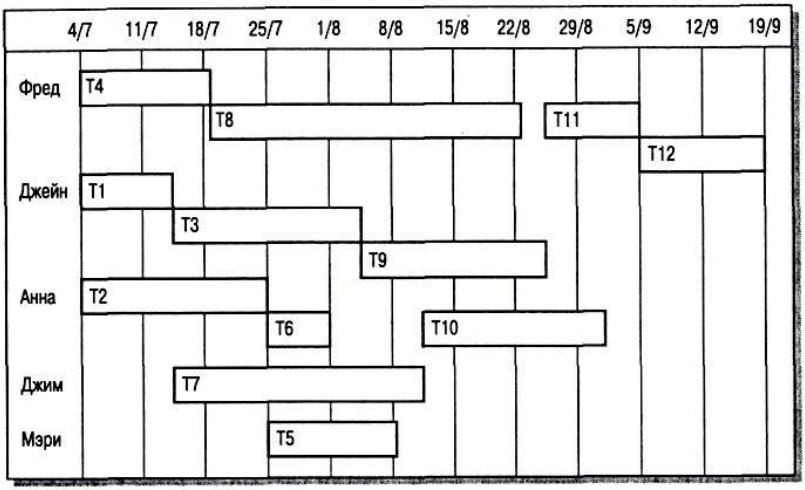

Рисунок 4 — Временная диаграмма распределения работников по этапам

В больших организациях обычно работает много специалистов, которые задействуются в проекте по мере необходимости. Конечно, такой подход может создать определенные проблемы для менеджеров проектов. Например, если специалист занят в

проекте, который задерживается, это может создать прямые сложности для других проектов, где он также должен участвовать.

Первоначальный график работ неизбежно содержит какие-нибудь ошибки или недоработки. По мере реализации проекта рассчитанные оценки длительности выполнения этапов работ должны сравниваться с реальными сроками выполнения этих этапов. Результаты сравнения должны использоваться в качестве основы для пересмотра графика работ еще не реализованных этапов проекта, в частности для того, чтобы попытаться уменьшить длительность этапов критического пути.

Методические рекомендации

Итак, на данном этапе необходимо провести декомпозицию работ, то есть разделить свой проект на подзадачи. Чем точнее будет детализирована работа, тем вероятнее, что вы ничего не забудете и не упустите из виду.

Затем составляется календарный план (Таблица 6). Временные оценки для каждой задачи строятся, как правило, на основании предыдущего опыта — данных о том, как долго подобного рода деятельность длилась в прошлом. Помните, что оценка действительна только для того человека, который будет выполнять работу. Каждую задачу, обозначенную в проекте, стоит поручить отдельному лицу.

Постройте сетевую модель, диаграмму Ганта и временную диаграмму распределения сотрудников по этапам (задачам). Для построения данных диаграмм желательно использовать специали-зированное программное обеспечении.

2.7 Проектирование программного обеспечения

В любой инженерной дисциплине под проектированием обычно понимается некий унифицированный подход, с помощью которого мы ищем пути решения определенной проблемы, обеспечивая выполнение поставленной задачи. По предположению Страуструпа: «Цель проектирования — выявление ясной и относительно простой внутренней структуры, иногда называемой архитектурой. Проект есть окончательный продукт процесса проектирования». Проектирование подразумевает учет противоречивых требований. Его продуктами являются модели, позволяющие нам понять структуру будущей системы, сбалансировать требования и наметить схему реализации.

Проектирование программного обеспечения — этап жизненного цикла разработки программного обеспечения, во время которого исследуется структура модулей программы и взаимосвязи элементов разрабатываемой системы.

Архитектура ПО — совокупность базовых концепций при по-строении программного обеспечения и связей между его компо-нентами.

На этапе проектирования ПО как правило определяется его структура, данные, которые являются частью системы , интерфейсы взаимодействия системных компонентов и иногда используемые алгоритмы. Процесс проектирования обычно проходит через разработку нескольких промежуточных версии ПО. Проектирование предполагает последовательную формализацию и детализацию создаваемого ПО с возможностью внесения изменений в решения, принятые на более ранних стадиях проектирования.

Конечными результатами процесса проектирования являются точные спецификации на алгоритмы и структуры данных, которые будут реализованы на следующем этапе создания ПО.

Ниже перечислены отдельные этапы процесса проектирования:

— *Архитектурное проектирование* — определяются и документируются подсистемы и взаимосвязи между ними.
— *Обобщенная спецификация* — для каждой подсистемы разрабатывается обобщенная спецификация на ее сервисы и ограничения.
— *Проектирование интерфейсов* — для каждой подсистемы определяется и документируется ее интерфейс. Спецификации на эти интерфейсы должны быть точно выраженными и однозначными, чтобы использование подсистем не требовало знаний о том, как они реализуют свои функции.
— *Компонентное проектирование* — проводится распределение системных функций (сервисов) по различным компонентам и их интерфейсам.
— *Проектирование структур данных* — детально разрабатываются структуры данных, необходимые для реализации программной системы.
— *Проектирование алгоритмов* — детально разрабатываются алгоритмы, предназначенные для реализации системных сервисов.

Рассмотрим основные используемые подходы при проектиро-вании программного обеспечения.

Нисходящее и восходящие проектирование и разработка

Методология проектирования играет наиважнейшую роль при построении правильных программных проектов. При разработке

программ на этапе проектирования обычно используется два под-хода: нисходящий и восходящий.

Восходящее проектирование (или проектирование «снизу вверх») основано на выделении нескольких достаточно крупных модулей, реализующих некоторые функции в общей программе. При выделении модулей опираются на доступность понимания реализуемых функций, простоту структурирования данных, существование готовых программ и модулей для реализации заданных функций, возможности переделки существующих программ для новых целей; имеет значение и размер будущего модуля. Каждый модуль при восходящем проектировании автономно программируется, тестируется и отлаживается. После этого отдельные моду-ли объединяются в подсистемы с помощью управляющего моду-ля, в котором определяется последовательность вызовов модулей, ввод-вывод и контроль данных и результатов. В свою очередь, подсистемы затем объединяются в более сложные системы и в общий программный комплекс, который подвергается комплексной отладке с проверкой правильности межмодульных связей. Основные недостатки восходящего проектирования программы проявляются в сложности объединения модулей в единую систему,
в трудности выявления и исправления ошибок, допущенных на ранних стадиях разработки модулей. Кроме того отдельные моду-ли могут создаваться без общего представления о структуре всей системы, что затрудняет их объединение.

Для создания сложных программ можно рекомендовать нисходящее проектирование, основанное на выделении в решаемой задаче иерархии уровней обобщения. Схема иерархии уровней обобщения позволяет программисту сначала сконцентрировать внимание на том, что нужно сделать, и лишь затем — на том, как это сделать. При нисходящей методике разработчик оценивает си-стему в целом, на уровне общей абстракции, формирует базовые классы, после чего «спускается» до разработки конкретных объ-ектов и специфики данной программы.

Сущность структурного подхода разработки программного обеспечения заключается в декомпозиции (разбиении) системы на функции, которые в свою очередь делятся на подфункции, на задачи и так далее. Процесс декомпозиции продолжается вплоть до определения конкретных процедур. При этом разрабатываемая система сохраняет целостное представление, в котором все со-ставляющие компоненты взаимосвязаны. В основу структурного подхода положены такие общие принципы:

— разбивка системы на множество независимых задач, легких для понимания и решения;

— иерархическое упорядочивание, т.е. организация составных частей проблемы в древовидные структуры с добавлением новых деталей на каждом уровне.

В основе этих принципов лежат операции:

— абстрагирования, то есть выделения существенных аспектов системы и отвлечения от несущественных;

— формализации, то есть строгое методологическое решение проблемы;

— непротиворечивости, состоящей в обосновании и согласовании элементов системы;

— структуризации данных (то есть данные должны быть структурированы и иерархически организованы).

При структурном анализе применяются в основном три вида наиболее распространённых моделей проектирования ПО:

— *SADT* (*Structured Analysis and Design Technique*) модель и соответствующие функциональные диаграммы;

— *SSADM (Structured Systems Analysis and Design Method)* — метод структурного анализа и проектирования;

— *IDEF0 (Integrated Definition Functions)* метод создания функциональной модели;

— *IDEF1* — информационной модели, *IDEF2* — динамической модели и др.

На стадии проектирования эти модели расширяются, уточняются и дополняются диаграммами, отражающими структуру программного обеспечения: архитектуру ПО, структурные схемы программ и диаграммы экранных форм.

Объектно–ориентированное проектирование (ООП) представляет собой стратегию, в рамках которой разработчики системы вместо операций и функций мыслят в понятиях объектов. *Объект* — это нечто, способное пребывать в различных состояниях и имеющее определенное множество операций. Состояние определяется как набор атрибутов объекта. Операции, связанные с объектом, предоставляют сервисы другим объектам (клиентам) для выполнения определенных вычислений. Объекты создаются в соответствии с определением класса объектов, который служит шаблоном для создания объектов. В него включены описания всех атрибутов и операций, связанных с объектом данного класса. Программная система состоит из взаимодействующих объектов, которые имеют собственное локальное состояние и могут выполнять

набор операций, определяемый состоянием объекта. Объекты скрывают информацию о представлении состояний и ограничивают к ним доступ. Под процессом объектно–ориентированного проектирования подразумевается проектирование классов объектов и взаимоотношений между этими классами.

Когда проект реализован в виде исполняемой программы, все необходимые объекты создаются динамически с помощью определений классов. Этот подход подразумевает выполнение трёх этапов при проектировании:

1. Объектно–ориентированный анализ. Создание объектно–ориентированной модели предметной области приложения. Здесь объекты отражают реальные объекты–сущности и операции, вы-полняемые этими объектами.

2. Объектно–ориентированное проектирование. Разработка объектно–ориентированной модели системы ПО (системной архитектуры) с учётом требований. В этой модели определение всех объектов подчинено решению конкретной задачи.

3. Объектно–ориентированное программирование. Реализация архитектуры (модели) системы с помощью объектно–ориентированного языка программирования (*C++, Java*) для определения объектов и средств определения классов объектов.

Объектно–ориентированные системы можно рассматривать как совокупность автономных и независимых объектов. Измене-ние реализации какого–нибудь объекта или добавление ему но-вых функций не влияет на другие объекты системы. Четкое соответствие между реальными объектами (например, аппаратными средствами) и управляющими объектами программной системы облегчает понимание и реализацию проекта.

Объекты могут быть повторно используемыми компонентами, они независимо инкапсулируют данные о состоянии и операци-ях. Архитектуру ПО можно разрабатывать на базе объектов, ранее созданных в предыдущих проектах. Это снижает стоимость проектирования, программирования и тестирования ПО. Кроме того, возможность использования стандартных объектов уменьшает риск, связанный с разработкой ПО.

Существует два типа моделей архитектуры ПО:

— статические модели, которые описывают статическую структуру системы в терминах классов объектов и взаимоотношений между ними. Основными взаимоотношениями, которые документируются на данном этапе, являются отношения обобщения, отношения «используют–используются» и структурные отношения.

— динамические модели, которые описывают динамическую структуру системы и показывают взаимодействия между объекта-ми системы (но не классами объектов).

Документируемые взаимодействия содержат последовательность запросов к сервисам объектов и описывают реакцию системы на взаимодействия между объектами.

Язык моделирования *UML (United Modeling Language* — унифицированный язык моделирования) поддерживает большое количество всевозможных статических и динамических моделей, в том числе — модель подсистем и модель последовательностей. Модель последовательностей — одна из наиболее полезных и наглядных моделей, которая в каждом узле взаимодействия документирует последовательность происходящих взаимодействий между объектами.

В *UML* используются следующие виды диаграмм:

Структурные диаграммы:

— Диаграмма классов (*Class diagram*)

— Диаграмма компонентов (*Component diagram*)

— Композитной/составной структуры (*Composite structure diagram*)

— Диаграмма кооперации (*Collaboration*) (*UML2.0*)

— Диаграмма развёртывания (*Deployment diagram*)

— Диаграмма объектов (*Object diagram*)

— Диаграмма пакетов (*Package diagram*)

— Диаграмма профилей (*Profile diagram*)(*UML2.2*)

Диаграммы поведения:

— Диаграмма деятельности (*Activity diagram*)

— Диаграмма состояний (*State Machine diagram*)

— Диаграмма прецедентов (*Use case diagram*)

— Диаграммы взаимодействия:

— Диаграмма коммуникации (*Communication diagram*) (*UML2.0*) / Диаграмма кооперации (*Collaboration*) (*UML1.x*)

— Диаграмма обзора взаимодействия (*Interaction overview diagram*) (*UML2.0*)

— Диаграмма последовательности (*Sequence diagram*)

— Диаграмма синхронизации (*Timing diagram*) (*UML2.0*)

Диаграмма классов (*Class diagram*) — статическая структурная диаграмма, описывающая структуру системы, она демонстрирует классы системы, их атрибуты, методы и зависимости между классами. Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

— концептуальная точка зрения — диаграмма классов описы-вает модель предметной области, в ней присутствуют только клас-сы прикладных объектов;

— точка зрения спецификации — диаграмма классов применяется при проектировании информационных систем;

— точка зрения реализации — диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программиро-вания).

Диаграмма компонентов (*Component diagram*) — статическая структурная диаграмма, показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонент могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т.п.

Диаграмма композитной/составной структуры (*Composite structure diagram*) — статическая структурная диаграмма, демонстрирует внутреннюю структуру классов и, по возможности, взаимодействие элементов (частей) внутренней структуры класса. Подвидом диаграмм композитной структуры являются диаграммы кооперации (*Collaboration diagram*, введены в *UML* 2.0), которые показывают роли и взаимодействие классов в рамках кооперации. Кооперации удобны при моделировании шаблонов проектирова-ния. Диаграммы композитной структуры могут использоваться совместно с диаграммами классов.

Диаграмма развёртывания (*Deployment diagram*) — служит для моделирования работающих узлов (аппаратных средств, англ. *node*) и артефактов, развёрнутых на них. Между артефактом и ло-гическим элементом (компонентом), который он реализует, уста-навливается зависимость манифестации.

Диаграмма объектов (*Object diagram*) — демонстрирует полный или частичный снимок моделируемой системы в заданный момент времени. На диаграмме объектов отображаются экземпля-ры классов (объекты) системы с указанием текущих значений их атрибутов и связей между объектами.

Диаграмма пакетов (*Package diagram*) — структурная диаграмма, основным содержанием которой являются пакеты и отношения между ними. Жёсткого разделения между разными структурными диаграммами не проводится, поэтому данное название предлагается исключительно для удобства и не имеет семантического значения (пакеты и диаграммы пакетов могут присутство-вать на других структурных диаграммах). Диаграммы пакетов

служат, в первую очередь, для организации элементов в группы по какому-либо признаку с целью упрощения структуры и органи-зации работы с моделью системы.

Диаграмма деятельности (*Activity diagram*) — диаграмма, на которой показано разложение некоторой деятельности на её составные части. Под деятельностью (англ. *activity*) понимается спецификация исполняемого поведения в виде координированно-го последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных дей-ствий (англ. *action*), соединённых между собой потоками, которые идут от выходов одного узла к входам другого. Диаграммы дея-тельности используются при моделировании бизнес-процессов, технологических процессов, последовательных и параллельных вычислений.

Диаграмма состояний (*State Machine diagram*, диаграмма конечного автомата) — диаграмма, на которой представлен конеч-ный автомат с простыми состояниями, переходами и композитны-ми состояниями. Конечный автомат (англ. *State machine*) — спецификация последовательности состояний, через которые проходит объект или взаимодействие в ответ на события своей жизни, а также ответные действия объекта на эти события. Конечный ав-томат прикреплён к исходному элементу (классу, кооперации или методу) и служит для определения поведения его экземпляров.

Диаграмма прецедентов (*Use case diagram,* диаграмма вариантов использования) — диаграмма, на которой отражены отношения, существующие между актёрами и вариантами использования. Основная задача — представлять собой единое средство, дающее возможность заказчику, конечному пользователю и разработчику совместно обсуждать функциональность и поведение системы.

Диаграмма коммуникации (*Communication diagram*, в *UML 1.x* — диаграмма кооперации, *collaboration diagram*) — диаграмма, на которой изображаются взаимодействия между частями ком-позитной структуры или ролями кооперации. В отличие от диа-граммы последовательности, на диаграмме коммуникации явно указываются отношения между элементами (объектами), а время как отдельное измерение не используется (применяются порядко-вые номера вызовов).

Диаграмма сотрудничества — этот тип диаграмм позволяет описать взаимодействия объектов, абстрагируясь от последовательности передачи сообщений. На этом типе диаграмм в компактном виде отражаются все принимаемые и передаваемые сообщения конкретного объекта и типы этих сообщений.

Диаграмма обзора взаимодействия (*Interaction overview diagram*) — разновидность диаграммы деятельности, включаю-щая фрагменты диаграммы последовательности и конструкции потока управления. Этот тип диаграмм включает в себя диаграм-мы *Sequence diagram* (диаграммы последовательностей действий) и *Collaboration diagram* (диаграммы сотрудничества). Эти диаграммы позволяют с разных точек зрения рассмотреть взаимодей-ствие объектов в создаваемой системе.

Диаграмма последовательности (*Sequence diagram*) — диаграмма, на которой изображено упорядоченное во времени взаимодействие объектов. В частности, на ней изображаются участву-ющие во взаимодействии объекты и последовательность сообще-ний, которыми они обмениваются.

Диаграмма синхронизации (*Timing diagram*) — альтернативное представление диаграммы последовательности, явным образом показывающее изменения состояния на линии жизни с задан-ной шкалой времени. Может быть полезна в приложениях реаль-ного времени.

Агентно-ориентированный подход (АОП) к программированию — разновидность представления программ, или парадигма программирования, в которой основополагающими концепциями являются понятия агента и его ментальное поведение, которое за-висит от среды, в которой он находится.

Агент — это самодостаточная программа, способная управлять своими действиями в информационной среде функционирования для получения результатов выполнения поставленной зада-чи и изменения текущего состояния среды.

Агент может обладать такими свойствами:

— автономность — это способность действовать без внешнего управляющего воздействия;

— реактивность — это способность реагировать на изменения данных и среды, и воспринимать их;

— активность — это способность ставить цели и выполнять заданные действия для достижения этой цели;

— социальность — это способность к взаимодействию с другими агентами (или людьми).

В задачи программного агента входят:

— самостоятельная работа и контроль своих действий;

— взаимодействие с другими агентами;

— изменение поведения в зависимости от состояния внешней среды;

— выдача достоверной информации о выполнении заданной функции и т. п.

С интеллектуальным агентом связываются знания типа: убеждение, намерение, обязательства и т.п. Эти понятия входят в концептуальную модель и связываются между собой операционны-ми планами реализации целей каждого агента. Для достижения целей интеллектуальные агенты взаимодействуют друг с другом, устанавливают связь между собой через сообщения или запросы и выполняют заданные действия или операции в соответствии с имеющимися знаниями. Агенты могут быть локальными и распределенными. Процессы локальных агентов протекают в клиент-ских серверах сети, выполняют заданные функции и не влияют на общее состояние среды функционирования. Распределенные агенты, расположенные в разных узлах сети, выполняют автоном-но (параллельно, синхронно, асинхронно) предназначенные им функции и могут влиять на общее состояние среды. В обоих слу-чаях характер взаимодействия между агентами зависит от таких факторов: совместимость целей, компетентность, не стандартные ситуации т.п.

Компонентный подход к проектированию

По оценкам экспертов, около 75 % работ по программирова-нию в мире дублируются (например, программы складского учета, начисления зарплаты, расчета затрат на производство продукции и т.п.). Большинство из этих программ типовые, но каждый раз на-ходятся особенности, которые влияют на их повторную разработ-ку. Компонентное проектирование сложных программ из готовых компонентов является наиболее производительным.

Компонентный подход дополняет и расширяет существующие подходы в программировании, особенно ООП (Объектно-ориентированное программирование). Объекты рассматриваются на логическом уровне проектирования программной системы, а компоненты — это непосредственная физическая реализация объ-ектов. Один компонент может быть реализацией нескольких объ-ектов или даже некоторой части объектной системы, полученной на уровне проектирования. Компоненты конструируются как не-которая абстракция, которая состоит из трех частей: информаци-онной, внешней и внутренней.

Информационная часть представляет собой информацию о компоненте: назначение, дата изготовления, условия применения (ОС, среда, платформа и т.п.); уровень повторного использования;

контекст или окружение; способ взаимодействия между собою компонентов.

Внешняя часть определяет взаимодействие компонента со средой и с платформой, на которой он будет выполняться. Эта часть имеет следующие основные характеристики:

— интероперабельность как способ взаимодействия с другими компонентами, с клиентом или сервером, а также обеспечения переносимости на другую платформу;

— способ интеграции (композиции) компонентов;

— нефункциональные сведения (безопасность, аутентификация, надежность и др.);

— технология проектирования (например, объектно-ориентированная среда и т.п.) и повторное использования компонентов.

Внутренняя часть — это некоторый артефакт (системная или абстрактная структура, фрагмент кода и др.) и вид его представле-ния: проектный компонент, проектная спецификация, вычисляе-мая часть, код и др.

Как правило, компоненты наследуются в виде классов. Управление компонентами проводится на трех уровнях: архитектурном, компонентном и на уровне инфраструктуры интерфейса.

Внутренняя часть компонента состоит из: интерфейса (*interfaces*), реализации (*implementation*), схемы развертки (*deployment*).

Интерфейсы отображают взгляд пользователя на компонент, то есть, что компонент будет делать, когда к нему обращаются.

Реализация — это код, который будет использоваться при обращении к операциям, которые определены в интерфейсах компонента. Компонент может иметь несколько реализаций, например, в зависимости от операционной среды или от модели данных и соответствующей системы управления базами данных, которая необходима для функционирования компонента.

Развертка — это физический файл или архив, готовый к выполнению, который передается пользователю и содержит все необходимые способы для создания, настройки и функционирования компонента.

Расширением понятия компонента есть *паттерн* — абстракция, которая содержит описание взаимодействия совокупности объектов в общей кооперативной деятельности, для которой определены роли участников и их ответственность. Представляется повторяемой частью программного элемента, как схемы или взаимосвязи контекста описания решения проблемы.

Повторно используемые компоненты (ПИК) — элементы знаний о минувшем опыте разработки систем, которые могут использовать не только их разработчиками, но и путём адаптации к новым условиям. ПИК упрощает и сокращает сроки разработки новых программных систем. Высокий уровень стандартизации и распространение электронных коммуникаций (сети Интернет) обеспечивает довольно простое получение и широкое использова-ние готовых компонентов в разных проектах за счет:

— отражения фундаментальных понятий приложения;

— скрытия способа представления и предоставления операций обновления и получения доступа;

— обработки исключительных операций в приложении.

Главным преимуществом создания программных систем из компонентов является уменьшение затрат на разработку за счет:

— выбора готовых компонентов с подобными функциями, пригодными для практического применения;

— настраивания готовых компонентов на новые условия, которые связаны с меньшими усилиями, чем разработка новых ком-понентов.

Методические рекомендации

Необходимо детально разработать структуру программы или вычислительной системы. Данная структура ПО должна отображать выделенные программные компоненты, а также интерфейсные свойства этих компонентов, отношения между ними, логику выполнения.

Разработанная архитектура ПО должна быть документирована. Документирование архитектуры ПО упрощает процесс коммуникации между заинтересованными лицами, позволяет зафиксировать принятые на ранних этапах проектирования решения о высокоуровневом дизайне системы, а также — повторно использовать компоненты и шаблоны этого дизайна в других проектах.

Выбранный вами подход к проектированию и модель архитек-туры необходимо описать, а также обосновать причины Вашего выбора.

2.8 Разработка программного обеспечения

Написание программ на скорую руку даёт быстрый, но сомнительный результат; дисциплинированный подход, напротив, обеспечивает более высокое качество с меньшими временными затратами.

Процесс реализации модулей можно разделить на несколько этапов:

1. Определить стандарты кодирования в рамках проекта.

2. Реализовать каждый класс каждого пакета через кодирование методов, определённых требованиями в детальном проектировании.

3. Осуществить инспектирование класса.

4. Произвести набор модульных тестов.

5. Выпустить пакет и классы для интеграции в разрабатываемое приложение.

Популярное представление о программировании как о процес-се подготовки текста программы для компиляции на самом деле отражает лишь незначительную часть общей картины. Действительной целью написания программы является создание правильного кода, то есть, такого кода, который полностью соответствует сформулированным для неё требованиям. Компиляторы только проверяют корректность синтаксиса и генерируют объектный код. В правильности программы нужно убедиться до начала компиляции, потому что за нее отвечает человек.

Рассмотрим один из способов реализации кода:

1. Спланируйте структуру вашего кода.
2. Проинспектируйте проект и структуру.
3. Наберите код:
   — пока не компилируйте;
   — попробуйте использовать методы, перечисленные ниже;
   — примените необходимые стандарты;
   — кодируйте таким способом, который будет легче всего проверить;
4. Проинспектируйте код, но пока не компилируйте.
5. Откомпилируйте код:
   — исправьте синтаксические ошибки.
6. Протестируйте код

Следование приведённому способу гарантирует написание качественного программного кода. Не менее важным аспектом является его читаемость при коллективной разработке программного обеспечения. Использование стандартов способствует дисципли-не программирования и повышает переносимость программ. Рассмотрим наиболее значимые моменты соглашения о кодировании. Указанные соглашения не являются строгим стандартом, но отражают генезис правил «чистого кодирования».

Соглашение об именах

1. Именовать объекты с помощью конкатенации слов, например *fileSize*. Такие имена легко понять, и они экономят место. Ис-

ключения со знаком подчёркивания оставляются на выбор программиста (вместо *aAAAAAAuto* лучше написать *a_AAA_AA_Auto*).

2. Начинать имена классов с большой буквы. Так вы будете легко отличать их от имён переменных. Иногда для именования классов проекта полезно выбрать префикс, чтобы отличать ваши классы от классов сторонних разработчиков (*PwUnit*, *ZnChannel*, *MwString*).

3. Имена переменных начинать с маленькой буквы.

4. Использовать в именах констант прописные буквы, например Я_ЕСТЬ_КОНСТАНТА (*CYCLE_REQUEST_TIME_OUT*, *CHANNEL_BOX_SIZE*).

5. Для методов доступа использовать имена, начинающиеся с *get*, *set* и *is*, например, *getName()*, *setName()*, *isBox()* — возвращает тип *boolean.*

6. Добавить стандартные дополнительные методы *g*et и *set* для коллекций, например *insertIntoName()*, *removeFromeName()*, *newName()*.

7. Договоритесь о соглашении для параметров. Можно использовать префиекс *a*, например *sum(int aNum1P, int aNum2P)*, или суффикс *P*, например *sum(int Num1P, int Num2P)*

Договоритесь о следующих стандартах внутри методов

1. выполняйте только одну операцию на одной строке

2. старайтесь чтобы ваши методы помещались на один экран

3. блоки в командах *if*, *else*, *while* должны состоять из одной строки в которой обычно содержится вызов функции.

4. для обеспечения принципов «защищённого» кодирования следует проверять все входные параметры на корректность.

Применяя этот минимальный набор умолчаний можно значительно улучшить восприятие вашего кода сторонними специалистами и подготовить его для повторного использования.

Методические указания

В рамках сформированных проектных групп провести собра-ние «соглашения о кодировании». На координирование соглаше-ний и способ их выработки следует выделить ограниченное время. Например, можно поручить члену команды разработать черновик соглашений и отправить его по электронной почте остальным членам команды для отзывов. Затем на собрании обсудить замечания и утвердить набор соглашений в форме документа.

В процессе написания кода осуществлять мониторинг чистоты и защищённости кода посредством инспектирования и модуль-ного тестирования.

2.9 Тестирование программного обеспечения

Тестирование — процесс исследования программного обеспечения с целью получения информации о качестве продукта. На начальных этапах ведения проекта следует отдавать себе отчёт в том, что мы не можем протестировать программу абсолютно во всех аспектах. Это обусловлено тем, что число инвариантов работы нетривиальной компьютерной программы может быть неограниченным. Следовательно, тестирование не может доказать отсутствия ошибок в программе, в то время как доказательство корректности способно это сделать.

Основной целью тестирования является точное определение мест неудовлетворительного функционирования программного продукта.

Тестирование оценивается более чем половиной времени, затраченного на проект. Наградой за нахождение дефекта на ранней стадии процесса является по крайней мере десятикратная эконо-мия по сравнению с обнаружением этого же дефекта на этапе ин-теграции или, ещё хуже, после отправки заказчику.

Тестирование программного обеспечения, как правило, включает в себя целый комплекс действий, аналогичный последовательности разработки ПО. В него входят:

— Определение целей тестирования, постановка задачи теста.

— Планирование тестирования — создание графика разработ-ки тестов, оценка человеческих, программных и аппаратных ре-сурсов, планирование проведения тестовых циклов.

— Проектирование, разработка и написание тестов.

— Выполнение тестов.

— Анализ результатов тестирования.

Выделяют три *уровня тестирования***:**

Модульное тестирование — процесс, позволяющий проверить на корректность отдельные модули исходного кода программы. Основной целью модульного программирования является изоля-ция отдельных частей программы и демонстрация работоспособ-ности этих частей (исследование структуры кода). Как правило, модульное тестирование осуществляется разработчиками ПО.

Интеграционное тестирование — процесс проверки корректного функционирования подсистем и интерфейсов между компонентами и подсистемами. Основная задача интеграционного тестирования — поиск дефектов, связанных с ошибками в реализации и интерпретации интерфейсного взаимодействия между модулями. Тестирование проводится итерационно, с постепенным подключением или отключение различных подсистем.

Системное тестирование — это тестирование, **выполняемое на полной**, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования чёрного ящика, и, тем самым, не требует знаний о внутреннем устройстве системы. Основной задачей системного тестирования является проверка как функциональных, так и нефункциональных требований к системе в целом. При этом выявляются такие дефекты, как: неверное использова-ние ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д. Для минимизации рисков, связанных с особенностями поведения системы в той или иной среде, во время тестирования рекомендует-ся использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи.

На этапе модульного тестирования широко распространены следующие подходы к проведению процесса тестирования:

— тестирование «*белого ящика*»,

— тестирование «*чёрного ящика*»,

— тестирование «*серого ящика*».

При тестировании «белого ящика» предполагается, что разработчик тестов имеет доступ к исходному коду и может писать код, связанный с библиотеками или подсистемами тестируемого ПО. Целью тестирования «белого ящика» — тестирование наиболее ненадёжных путей программы. Для его выполнения мы сначала разбиваем проект программы на отдельные элементы и ищем пути прослеживающие все или некоторые из этих путей, и проверяем все составные части.

Тестирование «серого ящика» рассматривает внутреннюю работу программы или модуля, но только до некоторой степени. Сюда могут быть также отнесены и некоторые аспекты тестирова-ния «чёрного ящика».

Когда мы интересуемся исключительно тем, как программа или её часть предоставляет соответствующие выходные данные, мы тестируем её на каждое требование, используя подходящие входные данные. Это называется тестированием «чёрного ящика». Тесты «чёрного ящика» могут быть эффективны, если мы можем убедиться, что они исчерпывают все комбинации входных дан-ных. Это докажет заказчику, что все требования удовлетворены. С практической точки зрения, для проведения тестирования по

этому методу, необходимо подготовить две группы входных условий . Одна группа должна содержать правильные входные данные для программы, вторая группа — неправильные, основанные на задании ошибочных входных значений. После прогона програм-мы на входных данных из обеих групп устанавливаются несоответствия между реальным поведением функций и ожидаемым.

К основным этапам тестирования в процессе интеграции мож-но отнести следующие:

— возможность повторного тестирования модулей в контексте системы (в пакетах);

— тестирование интерфейсов подразумевает валидацию интерфейсов между модулями;

— регрессионное тестирование — его цель заключается в проверке того, что добавление к системе новой функциональности не уменьшили её возможностей. Другими словами, регрессионное тестирование проводится согласно требованиям, которые уже были выполнены перед добавлением новых возможностей. Толь-ко когда артефакт прошёл регрессионное тестирование, мы будем готовы тестировать работу добавленного кода.

— интегральное тестирование выполняется над частично сконструированной системой для проверки того, что результат интеграции дополнительных программ работает, как запланировано.

— системное тестирование является кульминационным моментом интегрального тестирования. Основные свойства, которые подвергаются проверке при системном тестировании, перечислены ниже:

1) Объём — тестируемый продукт исследуется при подаче больших объёмов входных данных.

2) Удобство и простота использования — требуется измерить реакцию пользователя в любой заданной шкале.

3) Производительность — измерение скорости в различных условиях.

4) Конфигурация — настройка системы под разные аппаратные и программные архитектуры с фиксацией временных затрат на этот процесс.

5) Совместимость с другими обозначенными программными приложениями — измерение времени адаптации.

6) Надёжность и доступность — измерение периода работоспособного состояния в течение длительного времени.

7) Безопасность — среднее время взлома системы.

8) Использование ресурсов — измеряется степень использова-ния оперативной памяти, жёсткого диска и т.д.

9) Установка — программа устанавливается в разных условиях и измеряется время установки.

10) Восстанавливаемость — выполнение принудительных действий для зависания программы. Измеряется время восстанов-ления.

11) Удобство в эксплуатации — выполняется обслуживание программы в разных ситуациях. Измеряется время обслуживания.

12) Загрузка и стресс — программа подвергается экстремальному трафику данных и событий.

Выполнение тестов

Существует два основных подхода к выполнению тестов:

— ручное тестирование — оно выполняется человеком, который последовательно выполняет определённые действия, опреде-ляющие методику выполнения тестов и задающие порядок их вы-полнения;

— автоматизированное тестирование программного обеспече-ния — это процесс тестирования, который выполняется автомати-чески при помощи специализированных инструментов.

Существует два основных подхода к автоматизации тестирования: тестирование на уровне кода и *GUI*-тестирование (тестирование через графический пользовательский интерфейс, *GUI*-Graphical user interface). К первому типу относится, в частности, модульное тестирование. Ко второму — имитация действий поль-зователя с помощью специальных тестовых скриптов.

Инструментальные средства тестирования

Тестирование очень трудоёмкий и дорогой процесс, поэтому создан широкий спектр инструментальных средств, облегчающих и оптимизирующих процесс тестирования.

Инструмент для автоматизированного тестирования — это программное обеспечение, посредством которого осуществляется создание, отладка, выполнение и анализ результатов тестирова-ния. Данный вид тестирования становится важным механизмом проверки, гарантирующим точность и стабильность программ.

Выделяют следующие основные инструментальные средства тестирования:

1. *Организатор тестов*. Управляет выполнением тестов. Он отслеживает тестовые данные, ожидаемые результаты и тестируе-мые функции программы.

2. *Генератор тестовых данных*. Генерирует тестовые данные для тестируемой программы в заданных областях входных дан-ных.

3. *Оракул.* Генерирует ожидаемые результаты тестов. В качестве оракулов могут выступать предыдущие версии программы или исследуемого объекта.

4. *Компаратор файлов*. Сравнивает результаты тестирования с результатами предыдущего тестирования и составляет отчет об обнаруженных различиях.

5. *Генератор отчетов.* Формирует отчеты по результатам проведения тестов.

6. *Динамический анализатор.* Добавляет в программу код, который подсчитывает, сколько раз выполняется каждый оператор. После запуска теста создает исполняемый профиль, в котором показано, сколько раз в программе выполняется каждый оператор.

7. *Имитатор.* Их существует несколько типов: целевые имитаторы моделируют машину, на которой будет выполняться программа, а имитатор пользовательского интерфейса — это программа, управляемая сценариями, которая моделирует взаимодействия с интерфейсом пользователя; имитатор ввода-вывода генерирует последовательности повторяющихся транзакций*Методические рекомендации*

Силами проектных групп требуется разработать наборы модульных и интеграционных тестов. Результатом прохождения этапа тестирования программного продукта должны быть журналы обнаружения/исправления дефектов. Желательным является использование современных инструментальных средств тестирования.

2.10 Подготовка документации к проекту

Документирование ПО — это процесс создания документации на программу или программный продукт. *Документация на программное обеспечение* — это документы, сопровождающие программное обеспечение (ПО). Эти документы содержат сведения, необходимые для разработки, сопровождения и эксплуатации программного обеспечения.

Выделяют четыре основных типа документации на ПО:

— архитектурная/проектная — обзор программного обеспечения, включающий описание рабочей среды и принципов, которые должны быть использованы при создании ПО;

— техническая — документация на код, алгоритмы, интерфейсы, *API*;

— пользовательская — руководства для конечных пользовате-лей, администраторов системы и другого персонала;

— маркетинговая — рекламный буклет, который описывает, что делает продукт, его конкурентные преимущества и решения.

Документирование программного обеспечения осуществляет-ся в соответствии с Единой системой программной документации (ГОСТ 19.XXX). Так, ГОСТ 19.101—77 устанавливает виды про-граммных документов для программного обеспечения различных типов. Ниже перечислены основные программные документы по этому стандарту и указано, какую информацию они должны со-держать.

Спецификация должна содержать перечень и краткое описание назначения всех файлов программного обеспечения, в том числе, и файлов документации на него, и является обязательной для программных систем, а также их компонентов, имеющих самостоятельное применение.

Текст программы (код вида документа — 12) должен содержать текст программы с необходимыми комментариями. Необходимость этого документа определяется на этапе разработки и утверждения технического задания.

Описание программы (код вида документа — 13) должно содержать сведения о логической структуре и функционировании программы.

Ведомость эксплуатационных документов (код вида документа — 20) должна содержать перечень эксплуатационных документов на программу, к которым относятся документы с кодами 30, 31, 32, 33, 34, 35, 46. Необходимость этого документа также определяется на этапе разработки и утверждения технического за-дания.

Формуляр (код вида документа — 30) должен содержать основ-ные характеристики программного обеспечения, комплектность и сведения об эксплуатации программы.

Описание применения (код вида документа — 31) должно содержать сведения о назначении программного обеспечения, обла-сти применения, применяемых методах, классе решаемых задач, ограничениях для применения, минимальной конфигурации тех-нических средств.

Руководство системного программиста (код вида документа — 32) должно содержать сведения для проверки, обеспече-ния функционирования и настройки программы на условия конкретного применения.

Руководство программиста (код вида документа — 33) долж-но содержать сведения для эксплуатации программного обеспе-чения.

Руководство оператора (код вида документа — 34) должно содержать сведения для обеспечения процедуры общения опе-ратора с вычислительной системой в процессе выполнения про-граммного обеспечения.

Руководство по техническому обслуживанию (код вида доку-мента — 46) должно содержать сведения для применения тесто-вых и диагностических программ при обслуживании технических средств.

Программа и методика испытаний (код вида документа — 51) должны содержать требования, подлежащие проверке при испы-тании программного обеспечения, а также порядок и методы их контроля.

Пояснительная записка (код вида документа — 81) должна содержать информацию о структуре и конкретных компонентах программного обеспечения, в том числе схемы алгоритмов, их общее описание, а также обоснование принятых технических и технико-экономических решений. Составляется на стадии эскиз-ного и технического проектов.

При подготовке документации не следует забывать, что она разрабатывается для того, чтобы ее можно было использовать, и потому она должна содержать все необходимые сведения.

Методические рекомендации

Необходимо к разработанному программному продукту подго-товить руководство пользователя и руководство системного про-граммиста.

Руководство пользователя, как правило, содержит следую-щие разделы:

— общие сведения о программном продукте;

— описание установки;

— описание запуска;

— инструкции по работе (или описание пользовательского ин-терфейса);

— сообщения пользователю.

Раздел *Общие сведения о программе* обычно содержит наиме-нование программного продукта, краткое описание его функций, реализованных методов и возможных областей применения.

Раздел *Установка* обычно содержит подробное описание дей-ствий по установке программного продукта и сообщений, которые при этом могут быть получены.

В разделе *Запуск*, как правило, описаны действия по запуску программного продукта и сообщений, которые при этом могут быть получены.

Раздел *Инструкции по работе* обычно содержит описание ре-жимов работы, форматов ввода-вывода информации и возможных настроек.

Раздел *Сообщения пользователю* должен содержать перечень возможных сообщений, описание их содержания и действий, которые необходимо предпринять по этим сообщениям.

Руководство системного программиста должно содержать следующие разделы:

— общие сведения о программном продукте;
— структура;
— настройка;
— проверка;
— дополнительные возможности;
— сообщения системному программисту.

Раздел *Общие сведения о программе* должен включать описание назначения и функций программы, а также сведения о технических и программных средствах, обеспечивающих выполнение данной программы (например, объем оперативной памяти, требо-вания к составу и параметрам внешних устройств, требования к программному обеспечению и т. п.).

В разделе *Структура программы* должны быть приведены сведения о структуре программы, ее составных частях, о связях между составными частями и о связях с другими программами.

В разделе *Настройка программы* должно быть приведено описание действий по настройке программы на условия практического применения.

В разделе *Проверка программы* должно быть приведено описание способов проверки работоспособности программы, например контрольные примеры.

В разделе *Дополнительные возможности* должно быть приведено описание дополнительных возможностей программы и способов доступа к ним.

В разделе *Сообщения системному программисту* должны быть указаны тексты сообщений, выдаваемых в ходе выполнения настройки и проверки программы, а также в ходе ее выполнения, описание их содержания и действий, которые необходимо пред-принять по этим сообщениям.

3 Программные технологии управления проектами

3.1 GanttProject

GanttProject — является кросс-платформенным приложением с открытым исходным кодом.

Основным направлением его использования является планирование проектов и сопутствующие этому процессу задачи, в частности:

— построение диаграмм Ганта;

— распределение ресурсов по задачам;

— автоматическое построение *PERT* диаграмм на основе диаграмм Ганта;

— экспорт диаграмм в графический формат *PNG*, а также генерация *PDF* и *HTML* отчётов;

— импорт/экспорт диаграмм в *Microsoft Project*.

3.2 On-line сервисы управления проектами

За последние годы рынок *on-line* сервисов поддержки проек-тов значительно расширился. К наиболее популярным бесплатным средствам можно отнести: *BasecampHQ, Central Desktop, Wrike, Huddle, Comindwork, 5pm, Nozbe, LiquidPlanner, Zoho Projects, ProWorkFlow, AceProject, Projectplace, GoPlan, ProjectDesk* и т.д.

Можно выделить базовый набор функций, предоставляемых указанными сервисами:

— планирование проектов;

— управление задачами;

— совместное использование документов;

— управление контактами;

— средства мониторинга трудозатрат;

— средства коммуникации и уведомления;

— контроль версий;

— трэкеры ошибок и пожеланий;

— генераторы отчётов.

Можно отметить, что функциональности указанных ресурсов вполне достаточно для поддержки проектов среднего уровня сложности. Учитывая динамику развития *web*-технологий, рынок подобных продуктов будет со временем только расширяться.

3.3 Redmine

Redmine — довольно гибкая кросс-платформенная система управления проектами и отслеживания *ошибок*. Распространя-ется согласно *GNU General Public License. Redmine* написана на известном фреймворке *Ruby on Rails*. Как и большинство подоб-ных систем, позволяет расширять свою функциональность за счёт

сторонних плагинов. В базовом наборе (без установленных плаги-нов) продукт предоставляет следующие возможности:

— ведение нескольких проектов;

— гибкая система доступа, основанная на ролях;

— система отслеживания ошибок;

— диаграммы Ганта и календарь;

— ведение новостей проекта, документов и управление фай-лами;

— оповещение об изменениях с помощью RSS-потоков и элек-тронной почты;

— «*Wiki*» для каждого проекта;

— форумы для каждого проекта;

— учёт временных затрат;

— лёгкая интеграция с системами управления версиями (*SVN, CVS, Git, Mercurial, Bazaar* и *Darcs*);

— создание записей об ошибках на основе полученных писем;

— поддержка СУБД *MySQL, PostgreSQL, SQLite, Oracle*.

Является достаточно простым и в то же время достаточно функциональным продуктом.

3.4 Atlassian JIRA

JIRA — коммерческая система, предназначенная для организа-ции процесса контроля запросов и задач, имеющая часть функци-ональности больших систем управления проектами. Не является ориентированной на *IT* проекты, но легко к ним адаптируется за счёт широкого набора плагинов.

Основной задачей системы является учёт типизированных за-просов /задач с определённым набором полей. Допускается опи-сывать новые типы запросов (*Custom Issues*), помимо встроенных. Начальный набор запросов выглядит следующим образом:

— *Bug* (проблема мешающая функционированию продукта);

— *Improvement* (улучшение — не новая функциональность, а улучшение старой);

— *New Feature* (заказ на новую функциональность продукта);

— *Task* (задание, дело, задача, которые должны быть выпол-нены).

4 Рекомендации по выполнению курсовой работы

Рекомендуется следующий ход работы по процессу подготов-ки курсовой работы:

1. Формирование команды.

2. Выбор темы и согласование её с руководителем.

3. Изучение требований к оформлению работы.
4. Обсуждение целей и миссии проекта.
5. Выбор модели ЖЦ ПО.
6. Подбор литературы.
7. Разработка спецификаций ПО.
8. Проектирование.
9. Разработка.
10. Модульное и системное тестирование.
11. Подготовка документации к проекту.
12. Анализ полученных результатов.
13. Оформление отчёта о выполненном проекте.
14. Подготовка презентации проекта.
15. Защита курсовой работы.

4.1 Структура курсовой работы

Обязательными элементами курсовой работы являются:
1) титульный лист;
2) содержание;
3) введение;
4) теоретическая часть;
5) практическая часть;
6) заключение;
7) библиографический список;
8) приложения.

Титульный лист является первой страницей курсовой работы и должен содержать следующие сведения: наименование учреждения (учебного заведения), название (тему), сведения о выполнившем курсовую работу, сведения о руководителе, наименование места и год выполнения.

Образец титульного листа приведен в Приложении 1.

Содержание включает перечень основных элементов курсовой работы с указанием номеров страниц, с которых начинается их месторасположение.

Введение — своего рода «лицо» курсовой работы. Это форма-лизованный раздел, который должен содержать следующие руб-рики:

— *Актуальность темы исследования* — обоснование теоретической и практической важности выбранной темы исследования (выбранного проекта). Указывается наименование, возможности и характеристика области применения разрабатываемого программного продукта.

— *Цель и задачи курсовой работы* — краткая и четкая формулировка задачи**,** которую нужно выполнить для достижения цели курсовой работы.

— *Предмет исследования* — формулировка конкретного вопроса или анализируемой проблемы.

— *Структура работы* — краткая аннотация структуры и содержания работы. Например,: курсовая работа состоит из введения, 2 глав и заключения. В I главе (название) — рассматривается.... Во II главе (название) — анализируется... и даются рекомендации по.... В заключении приведены основные выводы, полученные в результате проведенного исследования.

Студент должен строго соблюдать приведенные здесь последовательность и названия рубрик, обязательно выделяя их в тексте введения жирным шрифтом.

Объём *Введения* — около 2 страниц.

Теоретическая часть должна содержать анализ предметной области, который позволяет выделить ее сущности, определить первоначальные требования к функциональности и определить границы проекта. Модель предметной области должна быть документирована. В данном разделе осуществляется постановка задачи, и предлагаются методы решение. Необходимо сделать обзор уже существующих программных систем аналогичного направления. В теоретической части даётся краткое описание инструментария, который применяется при разработке. Здесь приводится краткое описание среды разработки, используемых элементов управления, графических возможностей, средств связи с базой данных и других компонентов. В состав материала раздела может входить теоретический материал по процессу разработки программного обеспечения, на основе которого ведётся работа по проекту. Например, это может быть теоретический материал, включающий в себя основные определения и подходы по технологии разработ-ки программных продуктов, описание выбранной модели раз-работки программного обеспечения, теоретические аспекты де-композиции работ по проекту, анализ требований и определение спецификаций программного обеспечения и т.п. Таким образом, допускается включать в теоретическую часть всю необходимую информацию по разделам данного курса, по предметной области, по используемым информационным технологиям. А также жела-тельно привести описание используемого программного продукта по управлению вашим проектом.

Практическая часть описывает этапы решения задач, возникающих при практическом выполнении проекта. Описывается процесс проектирования, разработки и тестирования программно-го обеспечения. В состав материала раздела также входит описа-ние полученного результата. Практическая часть может содержать различный набор из нижеперечисленных пунктов:

1. Формулировку цели и миссии проекта.
2. Анализ требований и определение спецификаций к разраба-тываемому программному обеспечению.
3. Описание общей архитектуры системы.
4. Структурная или объектная схема разрабатываемого про-граммного продукта.
5. Схема информационных потоков.
6. Блок-схема взаимодействия различных компонентов про-граммы.
7. Необходимые виды *UML*-диаграмм.
8. Концептуальная модель и структурная схема базы данных.
9. Блок-схемы и описания применяемых алгоритмов.
10. Описание разработанного интерфейса.
11. Связь с базой данных и/или работа с файлами.
12. Описание структур и форматов данных в системе.
13. Описания классов.
14. Формы входных и выходных данных.
15. Описание методов тестирование и отладки программного продукта.
16. Описание документации к разработанному программному обеспечению.
17. Описание полученных результатов работы программы.

Заключение должно включать в себя основные выводы и результаты, перспективы использования и дальнейшего развития защищаемого программного обеспечения. Также перечисляются основные знания и навыки, полученные и усвоенные в ходе рабо-ты над курсовым проектом.

Список использованной литературы указывается в конце курсовой работы и составляется в алфавитном порядке. В основном тексте курсовой работы обязательны ссылки на все использованные источники. Список литературы включается в общую нумерацию курсовой работы.

Приложения помещаются в конце курсовой работы. Каждое приложение должно начинаться с новой страницы и иметь содержательный заголовок. Приложения нумеруются арабскими циф-

рами по порядковой нумерации. Номер приложения размещается в правом верхнем углу над заголовком приложения после слова «Приложение», после цифры точку не ставят. Приложения должны иметь общую с остальной частью курсовой работы нумерацию страниц. На все приложения в основной части курсовой работы должны быть ссылки.

4.2 Оформление курсовой работы Курсовая работа выполняется на стандартных листах формата А4 (210х297 мм). **Объем работы должен составлять 25-30 страниц** компьютерного текста, набранного шрифтом *Times New Roman* черного цвета с полуторным интервалом, высота букв, цифр и других знаков — не менее 1,8 мм (кегль равен 12). Полужирный шрифт не применяется. Абзацный отступ — 1,25 (5 знаков). Напечатанный текст должен иметь поля: верхнее — 20 мм, правое — 20 мм, левое — 20 мм, нижнее — 20 мм. Следует использовать режим выравнивания «по ширине».

Все листы работы нумеруются арабскими цифрами по порядку, включая приложения (номер указывается в центре нижнего поля без точки шрифтом № 12–14.). Первым листом считается титульный лист (Приложение А), номер на нем не проставляется.

Титульный лист должен содержать следующую информацию: название университета и кафедры, название дисциплины, наименование темы, сведения о студенте (Ф.И.О., курс, группа), руководитель (должность, Ф.И.О.), место и дата выполнения работы.

Слова «СОДЕРЖАНИЕ», «ВВЕДЕНИЕ», «ЗАКЛЮЧЕНИЕ», «СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ», «ПРИЛОЖЕ-НИЯ» размещается по центру страницы в виде заголовка пропис-ными буквами. Заголовки пунктов содержания (частей работы) записывают с прописной буквы строчными буквами. Заголовки пунктов содержания основной части работы должны иметь по-рядковую нумерацию в пределах каждого пункта, и обозначать-ся арабскими цифрами, например: 3.2.1.1, 3.2.1.2 и т.д. Заголовки подразделов записываются в центре строки (выравнивание — по центру) шрифтом *Times New Roman Cyr* № 16. Введение и заклю-чение не нумеруются.

Заголовки пунктов внутри подразделов записываются шрифтом *Times New Roman Cyr* № 14. Выравнивание — по центру. Межстрочный интервал — 1,5. Отступ красной строки — 12,5 мм. Заголовки в тексте и оглавлении должны совпадать.

В тексте работы могут быть перечисления. Перед каждой позицией перечисления следует ставить дефис или, при необходи-

мости, ссылки на одно из перечислений, строчную букву, после которой ставится скобка (без точки). Если необходима дальнейшая детализация перечислений, используют арабские цифры, после которых ставится скобка, а запись производится с абзацного отступа. Каждый пункт, подпункт и перечисление следует записы-вать с абзацного отступа.

После знаков препинания: точка, запятая, двоеточие, вопросительный и восклицательный знак — ставится пробел (отбивка); тире отбивается с двух сторон; дефис не отбивается; внутри ско-бок и кавычек пробел не ставится;

В тексте работы не допускается оставлять на листе первые и последние строки абзацев.

Разрешается использовать компьютерные возможности акцен-тирования внимания на определенных терминах, формулах, тео-ремах, применяя шрифты разной гарнитуры.

Отчёт по курсовой работе готовится в одном экземпляре и должен быть переплетён для сдачи её на проверку руководителю.

Формулы

При создании математических формул необходимо использовать редактор формул. Во всех формулах в качестве символов следует применять обозначения, определенные соответствующими государственными стандартами, или установившиеся в соответствующей профессиональной среде. Пояснения символов и числовых коэффициентов, входящих в формулу, если они не поясне-ны ранее в тексте, должны быть приведены непосредственно под формулой. Пояснения каждого символа следует давать с новой строки в той последовательности, в которой символы приведены в формуле. Первая строка пояснения должна начинаться со слова «где» без двоеточия после него.

Пример — Плотность каждого образца p, кг/м$_3$, вычисляют по формуле

$$p = \frac{m}{V} \quad , \qquad (1)$$

где m — масса образца, кг;

V — объем образца, м$_3$.

Перенос формулы на следующую строку допускается только на знаках выполняемых операций , причем знак в начале следующей строки повторяют. Формулы, следующие одна за другой и не разделенные текстом, разделяют запятой.

Все формулы в основном тексте работы должны нумероваться сквозной нумерацией арабскими цифрами, которые записывают на уровне формулы справа в круглых скобках. Номер формулы проставляется в круглых скобках арабскими цифрами с право-го края листа на уровне оси, проходящей через центр формулы. Сама формула должна быть отцентрована относительно текста. Ссылки в тексте на соответствующую формулу даются также в круглых скобках, например «...расчет данных проводился по фор-муле (1)...».

Формулы, помещаемые в приложениях, должны нумероваться отдельной нумерацией арабскими цифрами в пределах каждого приложения с добавлением перед каждой цифрой обозначения приложения, например формула (В.1).

Рисунки

Количество иллюстраций должно быть достаточным для пояс-нения излагаемого текста. Иллюстрации могут быть расположены как по тексту документа (возможно ближе к соответствующим ча-стям текста), так и в конце его.

К иллюстрациям относятся рисунки, схемы, диаграммы, графики и т.д. Рисунки следует нумеровать арабскими цифрами сквозной нумерацией. Каждый рисунок должен быть подписан, при этом используется слово «Рисунок». Слово «Рисунок» и наименование располагают посередине строки без кавычек, например, следующим образом: *Рисунок 2 — Форма заполнения анкеты.* В конце наименования рисунка точку не ставят. Сокращение слова «Рисунок» не допускается.

При ссылках на иллюстрации следует писать «... в соответ-ствии с рисунком 2» при сквозной нумерации и «... в соответствии с рисунком 1.2» при нумерации в пределах раздела. Иллюстрации каждого приложения обозначают отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения. Например — Рисунок А.3.

Таблицы

Таблицы применяют для лучшей наглядности и удобства сравнения показателей. Название таблицы, при его наличии, должно отражать ее содержание, быть точным, кратким. Таблицы следу-ет нумеровать арабскими цифрами сквозной нумерацией. Номер таблицы следует помещать над таблицей слева без абзацного отступа. Название таблицы располагают в одну строчку с ее номе-ром через тире. Например: Таблица 1 — Исходные данные.

Таблица — (номер) (название таблицы)

При переносе части таблицы на ту же или другие страницы название помещают только над первой частью таблицы.

Таблицы каждого приложения обозначают отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения. Если в документе одна таблица, она должна быть обозначена «Таблица 1» или «Таблица В.1», если она приведена в приложении В.

На все таблицы документа должны быть приведены ссылки в тексте документа, при ссылке следует писать слово «таблица» с указанием ее номера.

Библиографический список

Список использованной литературы помещается после основного текста курсовой работы и позволяет автору документально подтвердить достоверность и точность приводимого исследова-ния, он также отражает глубину и широту изучения темы, демонстрирует эрудицию и культуру исследователя.

Ссылка на источник в тексте работы приводится в квадратных скобках с указанием номера из списка литературы, например: «...в учебнике [2] дается такое определение...». Допускается также при ссылке на источник указывать соответствующую страницу издания, которая цитируется в работе: «...в учебнике [2, с. 12] дает-ся такое определение ...». Недопустимо заимствование текста из литературных источников без ссылки на автора цитаты. Также можно ссылаться на источники Интернет. На все источники, указанные в библиографическом списке, обязательно должны быть ссылки в тексте работы. В библиографическом списке должно быть не менее 15 источников.

Например:

Книга одного, двух, трёх авторов

1. Соммервилл, Иван. Инженерия программного обеспечения, 6-е издание. : Пер. с англ. — М.: Издательским дом «Вильямс», 2002. — 624 с.

2. Гаракина, Л. Г. Технология разработки программного обеспечения / Е. В. Кокорева, Б. Д. Виснадуп — М: ИД «ФОРУМ»: ИНФРА-М, 2009. — 400 с.

Книга четырёх и более авторов

1. Программирование на Microsoft Visual C++ для профессионалов : [пер. с англ.] / Д. Д. Круглински [и др.] — СПб.: Питер, 2002. — 864 с.

Источники Интернет

Здесь обязательно надо указывать название сайта и название статьи, на которую производится ссылка:

1. Разработка программного обеспечения. — Свободная энци-клопедия.

2. Фрагмент онтологии физической химии и его мо-дель / Ф. Ф. Иванов // Электрон. журн. — «Исследовано в Рос-сии», 10-14, 2008, 3. — Электрон. дан. — Режим доступа: http://zhumal.ape.relam.ru/articles/ 1998/003.pdf

Приложения

В приложение помещают материалы, дополняющие текст документа. Например, фрагменты программного кода курсовой работы, графики, таблицы, диаграммы с результатами и т.п.

Приложения размещают после списка использованных литературных источников и последовательно нумеруются заглавны-ми буквами русского алфавита (Приложение А, Приложение Б и т.д.). В тексте работы на все приложения должны быть приведены ссылки. Расположение приложений в конце документа должно соответствовать порядку появления ссылок на них в тексте.

Каждое приложение начинается с новой страницы. В верхнем правом углу страницы указывается слово «Приложение» и ставит-ся его порядковый номер (например, «Приложение В»). Каждое приложение должно иметь заголовок, который ставится на сле-дующей строке после слова «Приложение», и этот заголовок цен-труется относительно текста.

Приложение должно иметь общую с остальной частью рабо-ты сквозную нумерацию страниц. Все приложения должны быть перечислены в содержании с указанием их номеров и заголовков.

4.3 Рекомендации по разработке презентации По теме курсовой работы должна быть подготовлена электронная презентация, раскрывающая основное содержание, тему работы и достигнутые результаты.

Основная цель презентации курсовой работы состоит в том, чтобы студенты научились в наглядной форме представлять и защищать свои проекты, отстаивать актуальность проведённого исследования, перспективность разработанного программного обеспечения. Презентация должна выгодно подчеркивать профес-

сиональность студента, позволяя более наглядно и информативно донести необходимую информацию до слушателей. Правильно составленная презентация дает возможность за короткий проме-жуток времени вникнуть в суть изучаемых проблем и способов их решения.

Рекомендуемая структура презентации:

1. Титульный слайд. На данном слайде следует указать следующую информацию: название учебного заведения; тему курсовой работы; ФИО студента.

2. Введение. На данном слайде описывается предметная область и предпосылки к исследованию (разработке программного обеспечения), также обосновывается актуальность и необходимость данной работы (1-2 слайда).

3. Постановка задачи.

4. Известные ранее результаты и проблемы.

5. Предлагаемое Вами решение.

6. Используемый инструментарий.

7. Слайды, отражающие содержание работы и процесс исследования (4-8 слайдов).

8. Внедрения. Данный слайд необходим только для тех сту-дентов, которые имеют акты внедрения в промышленную или опытную эксплуатацию.

9. Основные результаты. На данном слайде следует поместить краткую сводку всех основных результатов, полученных в ходе исследования.

10. Наконец, последним слайдом должен быть лист со сло-вами «Доклад окончен. Спасибо за внимание».

Количество слайдов должно быть примерно 13-

18. Основные рекомендации:

— При подборе оформления презентации, следует учитывать, что демонстрация слайдов проводится на большом экра-не, поэтому фон слайдов следует делать либо тёмным с белым цветом текста, либо светлым с темным цветом текста.

— Перегруженность и мелкий шрифт в презентации тяжелы для восприятия, поэтому оптимальное число строк на слайде — от 6 до 11.

— Пункты и подпункты должны содержать одну, две стро-ки на фразу, Чтение длинной фразы отвлекает внимание от речи. Короткая фраза легче запоминается визуально.

— Текст желательно чередовать с интересными изображениями, наглядными примерами, диаграммами и графиками и т.п.

— В презентации не допускается разнобой в шрифтах и отступах, орфографические ошибки.

— Необходимо продумывать каждый слайд (что будет на слайде? что будет говориться? как будет сделан переход к сле-дующему слайду?).

— Каждая фраза в презентации также должна быть продумана, чтобы у слушателей не возникали проблемы с её восприятием. Если вы используете какие-либо термины, их непременно следует пояснять, хотя бы устно. Непонятные фразы не следует включать в презентацию.

— Желательно подготовить и отрепетировать речь выступления. Читать слайды презентации дословно не допускается. Должен быть лаконичный переход между слайдами.

— Оптимальная скорость переключения — один слайд за 1–2 минуты. Слушатели должны успеть воспринять информа-цию и со слайда, и на слух.

— На слайдах с ключевыми определениями можно задержаться подольше. Если они не будут поняты, то не будет понято ничего.

Для демонстрации работы программ в презентации можно использовать скриншоты или анимационные ролики.

Время доклада — 15 мин., плюс 5 мин. для ответов на вопросы.

Не надо думать, что хорошую презентацию можно сделать за три часа накануне выступления. Это грозит провалом. Уделите подготовке презентации и выступления 1-2 недели.

Библиографический список

1. Соммервилл, Иван. Инженерия программного обеспечения, 6-е издание / Иван Соммервилл Пер. с англ. — М.: Издательским дом «Вильямс», 2002. — 624 с.
2. Гагарина, Л. Г. Технология разработки программного обеспече-ния / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадуп — М: ИД «ФОРУМ»: ИНФРА-М, 2009. — 400 с.
3. Лаврищева Е. М., Петрухин В. А. Методы и средства инженерии программного обеспечения: Учебник. / Е. М. Лаврищева, В. А. Петрухин. — М.: МФТИ (ГУ), 2006. — 304 с.
4. Программирование на Microsoft Visual C++ для профессионалов : [пер. с англ.] / Д. Д. Круглински [и др.] — СПб.: Питер, 2002. — 864 с.
5. Шеферд, Дж. Программирование на *Microsoft Visual Studio C++.NET* : [пер. с англ.] / Дж. Шеферд. — М.: Издательство «Русская редакция»; СПб.: Питер, 2007. — 928 с.
6. Грищенко В. Н., Лаврищева Е. М. Методы и средства компонентного программирования//Кибернетика и системный анализ, 2003. — №1. — с. 39–55.

Приложения

Приложение А Образец титульного листа курсовой работы

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ
Филиал «Протвино» государственного университета «Дубна»

Филиал «Протвино»
Кафедра «Информационные технологии»
(наименование кафедры)

КУРСОВАЯ РАБОТА
ПО

(наименование учебной дисциплины)

ТЕМА:
(наименование темы)

Выполнил: студент
_____ группы
_____ курса

(Ф.И.О.)

Руководитель:

(ученая степень, ученое звание, занимаемая должность)

Дата защиты: _____
Оценка: _____

(подпись руководителя)

Электронное учебное издание

Ковцова Ирина Олеговна
Мандрик Андрей Владимирович
Ухов Владимир Иосифович

Подготовка курсовых работ по дисциплине «Технология разработки программного обеспечения»

ЭЛЕКТРОННОЕ МЕТОДИЧЕСКОЕ ПОСОБИЕ

Филиал «Протвино»
государственного университета «Дубна
142281 г. Протвино Московской обл.,
Северный проезд, 9