Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Северо-Кавказский федеральный университет»
Колледж СКФУ в г. Ставрополе

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

ПМ.03 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНФОРМАЦИОННЫХ СИСТЕМ

Специальность 09.02.11 Разработка и управление программным обеспечением

Форма обучения очная

Ставрополь

Методические указания для учебной дисциплины разработаны на основании федерального государственного образовательного стандарта среднего профессионального образования по специальности 09.02.11 Разработка и управление программным обеспечением и примерной основной образовательной программы СПО, с учетом направленности на удовлетворение потребностей регионального рынка труда и работодателей.

Целью выполнения практических занятий является систематизация и закрепление теоретических знаний и формирование практических умений.

Особое значение для усвоения практических навыков имеет правильная и четкая организация проведения и выполнения студентами практических работ под контролем преподавателя.

Перед началом выполнения каждой работы студенты должны ознакомиться с ее основными положениями, порядком выполнения работы.

По каждому практическому занятию предусматривается индивидуальный отчет перед преподавателями.

Выполнение студентами практических работ направленно на:

- обобщение, систематизацию, углубление, закрепление полученных теоретических знаний по конкретным темам;
- формирование умений применять полученные знания на практике и реализацию единства интеллектуальной и практической деятельности;
- развитие интеллектуальных умений у будущих специалистов: аналитических, проектировочных, конструктивных и др.;
- выработку при решении поставленных задач таких профессиональных качеств, как самостоятельность, ответственность, точность, творческая инициатива.

В результате изучения профессионального модуля обучающийся должен уметь:

- осуществлять постановку задачи по обработке информации. Выполнять анализ предметной области;
- использовать алгоритмы обработки информации для различных приложений;
- работать с инструментальными средствами обработки информации;
- осуществлять выбор модели построения информационной системы;
- осуществлять выбор модели и средства построения информационной системы и программных средств;
- осуществлять математическую и информационную постановку задач по обработке информации;
- использовать алгоритмы обработки информации для различных приложений;
- осуществлять математическую и информационную постановку задач по обработке информации;
- использовать алгоритмы обработки информации для различных приложений;
- создавать и управлять проектом по разработке приложения и формулировать его задачи;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;
- разрабатывать графический интерфейс приложения;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ.
- решать прикладные вопросы программирования и языка сценариев для создания программ;
- проектировать и разрабатывать систему по заданным требованиям и спецификациям;
- разрабатывать графический интерфейс приложения;
- создавать проект по разработке приложения и формулировать его задачи;

- использовать методы тестирования в соответствии с техническим заданием;

- разрабатывать проектную документацию на эксплуатацию информационной системы;

- использовать стандарты при оформлении программной документации;

- использовать методы и критерии оценивания предметной области и методы определения

стратегии развития бизнес- процессов организации.

- решать прикладные вопросы интеллектуальных систем с использованием статических экспертных систем, экспертных систем реального времени.

В результате освоения профессионального модуля обучающийся должен знать:

- основные виды и процедуры обработки информации, модели и методы решения задач обработки информации;

- основные платформы для создания, исполнения и управления информационной системой;

- основные модели построения информационных систем, их структуру, особенности и области применения.

- платформы для создания, исполнения и управления информационной системой;

- основные процессы управления проектом разработки.

- методы и средства проектирования, разработки и тестирования информационных систем;

- основные платформы для создания, исполнения и управления информационной системой;

- национальную и международную систему стандартизации и сертификации и систему обеспечения качества продукции, методы контроля качества;

- сервисно - ориентированные архитектуры.

- важность рассмотрения всех возможных вариантов и получения наилучшего решения на основе анализа и интересов клиента;

- методы и средства проектирования информационных систем;

- основные понятия системного анализ;

- национальной и международной системы стандартизации и сертификации и систему обеспечения качества продукции;

- методы контроля качества объектно-ориентированного программирования.

- спецификации языка программирования, принципы создания графического пользовательского интерфейса (GUI), файлового ввода-вывода, создания сетевого сервера и сетевого клиента;

- файлового ввода-вывода. Создания сетевого сервера и сетевого клиента;

- особенности программных средств, используемых в разработке ИС;

- основные модели построения информационных систем, их структура;

- реинжиниринг бизнес-процессов;

- системы обеспечения качества продукции;

- методы контроля качества в соответствии со стандартами.

Правила выполнения практической работы.

1. Студент должен прийти на лабораторную работу подготовленным к выполнению лабораторной работы.
2. Каждый студент после проведения работы должен представить отчет о проделанной работе с анализом полученных результатов и выводом по работе.
3. Таблицы и рисунки следует выполнять с помощью чертежных инструментов (линейки, циркуля, и т.д.) карандашом с соблюдением ЕСКД.
4. Расчет следует проводить с точностью до двух значащих цифр.
5. Исправления проводить на обратной стороне листа. При мелких исправлениях неправильное слово (буква, число и т.п.) аккуратно зачеркивается и над ним пишут правильное пропущенное слово (букву, число и т.п.).
6. Вспомогательные расчеты можно выполнять на отдельных листах, а при необходимости на листах отчета.
7. Если студент не выполнит лабораторную работу или часть работы, то он выполнит ее во внеурочное время, согласованное с преподавателем.
8. Оценку по лабораторной работе студент получает с учетом срока выполнения работы,

если;

- расчеты выполнены правильно и в полном объеме;
- сделан анализ проделанной работы и вывод по результатам работы;
- студент может пояснить выполнение любого этапа работы;
- отчет выполнен в соответствии с требованиями к выполнению работы.

Раздел. Тестирование информационных систем

Тема. Основные этапы и методологии в проектировании и внедрении информационных систем

Практическая работа №1. Разработка технического задания на внедрение информационной системы.

Цель работы: освоение технологии документирования программных средств на начальных стадиях проектирования ИС в соответствии Сеспд

Форма отчета:

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

1. Краткие теоретические сведения. Основу отечественной нормативной базы в области документирования ПС составляет комплекс стандартов Единой системы программной документации (ЕСПД).

Основная и большая часть комплекса ЕСПД была разработана в 70-е и 80-егоды 20 века. Сейчас этот комплекс представляет собой систему межгосударственных стандартов стран СНГ (ГОСТ), действующих на территории Российской Федерации на основе межгосударственного соглашения по стандартизации.

Единая система программной документации - это комплекс государственных стандартов, устанавливающих взаимоувязанные правила разработки, оформления и обращения программ и программной документации. Стандарты ЕСПД в основном охватывают ту часть документации, которая создается в процессе разработки программных средств, и связаны, по большей части, с документированием функциональных характеристик программных средств. Следует отметить, что стандарты ЕСПД (ГОСТ 19) носят рекомендательный характер.

Впрочем, это относится и ко всем другим стандартам в области ПС (ГОСТ34, международному стандарту ISO/IEC и др.). Дело в том, что в соответствии с Законом РФ «О стандартизации» эти стандарты становятся обязательными на контрактной основе, т.е. при ссылке на них в договоре на разработку (поставку)программного средства.

Говоря о состоянии ЕСПД в целом, можно констатировать, что большая часть стандартов ЕСПД морально устарела. Тем не менее до пересмотра всего комплекса многие стандарты могут с пользой применяться в практике документирования программных средств. К числу программных ЕСПД относит документы, содержащие сведения, необходимые для разработки, изготовления, сопровождения и эксплуатации программ. Как известно, грамотно составленный пакет программной документации позволяет избежать при проектировании многих неприятностей. В частности, избавиться от назойливых вопросов и необоснованных претензий заказчика можно, просто отослав пользователя к документации.

Это касается прежде всего важнейшего документа — Технического задания.

Техническое задание (ТЗ) содержит совокупность требований к программному средству и может использоваться как критерий проверки и приемки разработанной программы. Поэтому достаточно полно составленное с учетом возможности внесения дополнительных разделов) и принятое заказчиком и разработчиком ТЗ является одним из основополагающих документов проекта программного средства.

ГОСТ 19.201-78, входящий в ЕСПД, устанавливает порядок построения и оформления технического задания на разработку программы или программного изделия для вычислительных машин, комплексов и систем независимо от их назначения и области

применения.

2. Задание: разработать техническое задание на проектирование информационной системы, предназначенной для решения задач автоматизации деятельности организации. Исходными данными для проектирования информационной системы являются описание предметной области и виды запросов в информационной системе(приложение 1).

Алгоритм выполнения работы

В соответствии с назначенным преподавателем вариантом определить наименование информационной системы (табл. 1), подлежащей проектированию входе лабораторного практикума, для удовлетворения основных требований к ней с применением системы управления базами данных Microsoft Access 2007 и/или инструментального средства Borland Turbo Delphi.

№ варианта Наименование информационной системы

1 Информационная система медицинских организаций города

2 Информационная система автопредприятия города

3 Информационная система проектной организации

4 Информационная система ГИБДД

5 Информационная система строительной организации

6 Информационная система библиотечного фонда города

7Информационная система спортивных организаций города

8 Информационная система аэропорта

9 Информационная система гостиничного комплекса

10 Информационная система торговой организации

11 Информационная система ВУЗа

12 Информационная система железнодорожной пассажирской станции

13 Информационная система зоопарка

14 Информационная система театра

15 Информационная система фотоцентра

2) Изучить описание предметной области информационной системы.

3. На основании анализа описания предметной области и запросов к будущей информационной системе сформулировать основные требования кее функциям.

4. Выполнить поиск прототипа проектируемой информационной системы с применением Интернет.

5. Используя сформулированные требования к информационной системе, а также документацию пользователя на прототип найденного программного средства, разработать техническое задание в соответствии с ГОСТ 19.201-78.

Практическая работа №2. Описание тестируемой системы.

Практикум базируется на тестировании модели реальной системы управления автоматизированным комплексом хранения подшипников. Она обеспечивает прием подшипников на склад, сохранение характеристик поступивших подшипников в базе данных (БД), а при поступлении заявки на подшипники вместе с параметрами оси - подбор подходящих подшипников и их выдачу. У каждого из элементов комплекса (склада, терминала подшипника и терминала оси) существует программа низкоуровневого управления, реализованная в виде динамически подключаемой библиотеки (dll), принимающая на вход высокоуровневые команды, и преобразующая их в управляющие воздействия на данный элемент тестируемой системы. Таким образом, есть реальное

окружение - аппаратура и dll, которые осуществляют связь с аппаратурой. Система вызывает следующие функции из dll для своих элементов (см. рис. 1.1):

GetStoreStat, GetStoreMessage, SendStoreCom (Store.dll) для склада. GetAxlePar (Axle.dll) для терминала оси.

GetRollerPar (Bearing.dll) для терминала подшипника.

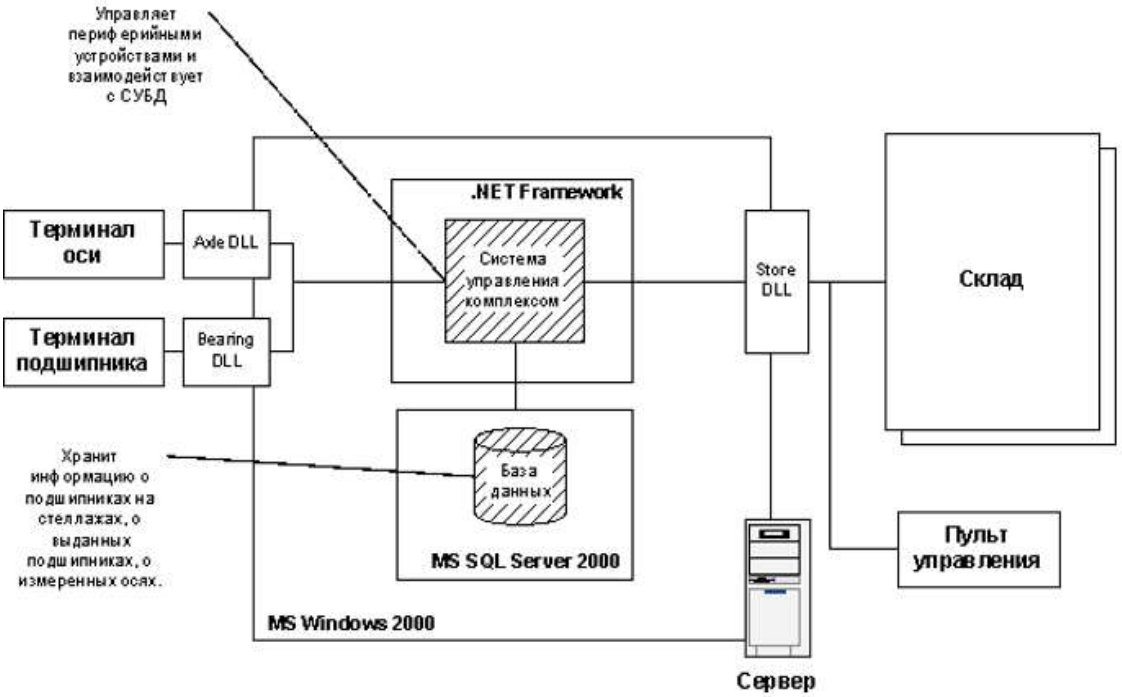


Рис. 1.1. Система и ее окружение

Поскольку для тестирования используется модель системы, в ее составе реальное окружение заменено на модельное, обеспечиваемое специальной библиотекой dll-функций окружения.

Тестируемая система реализована как многопоточное приложение. Многопоточность порождает недетерминированность поведения системы во времени. Поэтому при тестировании необходимо учитывать, что возможны различные варианты допустимых временных последовательностей событий системы. Кроме того, совсем не просто точно воспроизвести прогон конкретного теста, когда система содержит много параллельных потоков, так как планировщик операционной системы сам определяет порядок событий. Изменения, вносимые в программы, не связанные с тестируемой системой, могут повлиять на порядок, в котором будут выполняться (воспроизводиться) потоки событий тестируемой системы. Это может привести к тому, что после выявления и исправления дефекта при проведении повторного тестирования далеко не всегда удастся убедиться в том, что дефект действительно устранен, если ошибка не обнаружена во время прогона.

При системном тестировании мы рассматриваем систему как черный ящик. Тестовый случай (test case) представляет собой пару (входные данные, ожидаемый результат), в которой входные данные - это описание данных, подаваемых на вход нашей системы, а ожидаемый результат - это описание выходных данных, которые система должна предъявить в ответ на соответствующий ввод. Выполнение (прогон) тестового случая - это сеанс работы системы, в рамках которого на вход системы подаются наборы данных, предусмотренные спецификацией тестового случая, и фиксируются результаты их обработки, которые затем

сравниваются с ожидаемыми результатами, указанными в тестовом случае. Если фактический результат отличается от ожидаемого, значит, обнаружен отказ, т.е. тестируемая система не прошла испытание на заданном тестовом случае. Если полученный результат совпадает с ожидаемым, значит, тестируемая система прошла испытание на заданном тестовом случае. Из тестовых случаев формируются тестовые наборы (test suits). Тестовые наборы организованы в определенном порядке, отражающем свойства тестовых случаев. Если система успешно справилась со всеми тестовыми случаями из набора, то она успешно прошла испытания на тестовом наборе.

Для нашей системы входными данными является состояние окружения ее компонентов:

Склад. Состояние склада будет характеризоваться следующими

параметрами: Статус склада (StoreStat).

Сообщение от склада о результатах выполнения команды (StoreMessage).

Сообщение от склада о результатах получения команды - статус команды (CommandStatus).

Терминал подшипника. Состояние терминала подшипника задается следующими параметрами:

Статус обмена с терминалом подшипника.

Характеристики (параметры) подшипника (RollerPar):

ФИО мастера, производившего измерения.

Название депо.

Номер рабочей смены.

Номер подшипника.

Номер группы подшипника.

Тип сепаратора подшипника.

Терминал оси. Состояние терминала оси задается следующими параметрами:

Статус обмена с терминалом оси.

Характеристики (параметры) оси (AxlePar):

ФИО мастера, производившего измерения.

Название депо.

Номер оси.

Сторона оси: правая или

левая. Посадочный диаметр

задний. Посадочный

диаметр передний.

База данных (БД). В БД хранятся характеристики поступивших на склад подшипников. При выборе подходящего для оси подшипника система обращается за этой информацией к БД. Поэтому имеет смысл предварительно очистить БД или заполнить ее определенными данными.

В спецификации тестового случая должны быть заданы состояние окружения (входные данные) и ожидаемая последовательность событий в системе (ожидаемый результат). После прогона тестового случая мы получим реальную последовательность событий в системе (выходные данные) при заданном состоянии окружения. Сравнивая фактический результат и ожидаемый, можно сделать вывод о том, прошла ли тестируемая

система испытание на заданном тестовом случае. В качестве ожидаемого результата будем использовать пошаговое описание случая использования (use case), так как оно определяет, как при заданном состоянии окружения система должна функционировать. Задавая ожидаемый результат, очень важно помнить о том, что при заданном состоянии окружения возможны различные варианты последовательности событий системы, которые все являются правильными.

В процессе работы последовательность событий (команд) системы, или история системы, записывается в журнал (log) системы. Вы можете использовать SystemLogAnimator (см. п.14 SysLog Animator Manual) для визуализации журнала системы. Выбирая различные log-файлы системы для визуализации, можно получить наглядное и достаточно полное представление о функционировании системы и о том, какие события и в каком порядке могут происходить в системе.

Практическая работа №3. Разработка перечня обучающей документации на информационную систему.

Цель: знакомиться с процедурой разработки руководства пользователя на создание программного продукта (ПП) с применением РД50-34-698-90 «Автоматизированные системы требования к содержанию документов».

Задание: во время выполнения практического занятия необходимо составить документ «Руководство пользователя» к разработанной ранее программе. Работа должна быть оформлена в виде документа «Руководство пользователя» согласно РД50-34-698 90 «Автоматизированные системы требования к содержанию документов».

В практической работе можно использовать **структуру руководства пользователя**. Некоторые разделы, которые можно включить: lib.laop.ulstu.ru

- **Введение** — предоставляет пользователю общую информацию о системе: область применения, краткое описание возможностей, уровень подготовки пользователя.
- **Перечень эксплуатационной документации** — перечисляет документацию, которая позволит пользователю избежать определённого рода ошибок.
- **Назначение и условия применения** — подразделяет основную задачу системы на подзадачи и описывает каждую из них. Указывает виды деятельности, функции, для автоматизации которых предназначено средство автоматизации, и условия, при соблюдении которых обеспечивается применение системы в соответствии с назначением.
- **Подготовка к работе** — содержит пошаговую инструкцию для запуска системы. Указывает состав и содержание дистрибутивного носителя данных, порядок загрузки данных и программ.
- **Проверка работоспособности** — описывает показатели, по которым можно определить, что программное обеспечение работает нестабильно.

Практическая работа №4. Описание окружения тестируемой системы. Планирование тестирования .

Процесс тестирования

Процесс тестирования находится в прямой зависимости от процесса разработки программного обеспечения, но при этом сильно отличается от него, поскольку преследует другие цели. Разработка ориентирована на построение программного продукта, тогда как тестирование отвечает на вопрос, соответствует ли разрабатываемый программный продукт требованиям, в которых зафиксирован первоначальный замысел изделия (т.е. то, что заказал заказчик).

Вместе оба процесса охватывают виды деятельности, необходимые для получения качественного продукта. Ошибки могут быть привнесены на каждой стадии разработки.

Следовательно, каждому этапу разработки должен соответствовать этап тестирования. Отношения между этими процессами таковы, что если что-то разрабатывается, то оно подвергается тестированию, а результаты тестирования используются для определения, соответствует ли это "что-то" набору предъявляемых требований. Процесс тестирования возвращает выявленные им ошибки в процесс разработки. Процесс разработки передает процессу тестирования новые и исправленные проектные версии.

Планирование тестирования

Как было отмечено выше, процесс тестирования тесно связан с процессом разработки. Соответственно планирование тестирования тоже зависит от выбранной модели разработки. Однако вне зависимости от модели разработки при планировании тестирования необходимо ответить на пять вопросов, определяющих этот процесс:

Кто будет тестировать и на каких этапах?

Разработчики продукта, независимая группа тестировщиков или совместно?

Какие компоненты надо тестировать?

Будут ли подвергнуты тестированию все компоненты программного продукта или только компоненты, которые угрожают наибольшими потерями для всего проекта?

Когда надо тестировать?

Будет ли это непрерывный процесс, вид деятельности, выполняемый в специальных контрольных точках, или вид деятельности, выполняемый на завершающей стадии разработки?

Как надо тестировать?

Будет ли тестирование сосредоточено только на проверке того, что данный продукт должен выполнять, или также на том, как это реализовано?

В каком объеме тестировать?

Как определить, в достаточном ли объеме выполнено тестирование, или как распределить ограниченные ресурсы, выделенные под тестирование?

Кто будет тестировать?

Разработчик - это роль, для которой характерны виды деятельности, ориентированные на создание программного продукта (ПП). Тестировщик - это роль, для которой характерны виды деятельности, ориентированные на улучшение/обеспечение качества программного продукта. Эта роль предусматривает выбор тестов, необходимых для конкретных целей, построение тестов, выполнение тестов и оценку результатов. Конкретный исполнитель проекта может выступать как в роли разработчика, так и в роли тестировщика. Момент начала тестирования в проекте можно регулировать (рис. 1.2).

Рис. 1.2. Кто тестирует

В рамках данного практикума студенту предназначена роль тестировщика.

Какие компоненты надо тестировать?

Могут быть варианты, когда ничего не надо тестировать (поскольку все компоненты

были протестированы ранее), а может потребоваться тестировать каждый компонент с точностью до строки кода. В объектно- ориентированном программировании базовым компонентом является класс. В этом случае область тестирования определяется классами. Область тестирования на уровне классов подлежит выбору (рис. 1.3). Классы, заимствованные из других проектов или взятые из библиотек, чаще всего в повторном тестировании не нуждаются. Существуют различные стратегии по выбору подмножества классов для тестирования.

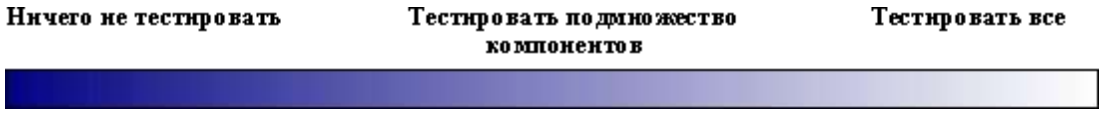


Рис. 1.3. Что тестировать

В нашем случае будет производиться тестирование всех классов приложения.

Когда надо тестировать?

Компоненты можно тестировать на завершающем этапе, когда они будут интегрированы в единый выполняемый модуль. Частота тестирования определяется различными соображениями. Можно проводить тестирование каждый день, учитывая тот факт, что чем раньше выявлена проблема, тем легче и дешевле ее решение. Можно тестировать программный компонент по мере завершения его разработки (рис. 1.4). Частое тестирование компонентов несколько замедляет ранние этапы разработки, однако сопряженные с этим потери с лихвой восполняются за счет меньшего числа проблем на более поздних этапах разработки проекта, когда отдельные модули объединяются в более крупные компоненты системы.

В случае, когда компоненты системы не отличаются большой сложностью, можно сначала осуществлять интегрирование не подвергавшихся автономному тестированию компонентов, а затем тестировать объединенный код как единое целое. Такой подход полезен при тестировании компонентов, для которых реализация тестовых драйверов требует существенных усилий. Тестовый драйвер представляет собой программу, которая выполняет прогон тестовых случаев и сбор полученных при этом результатов.

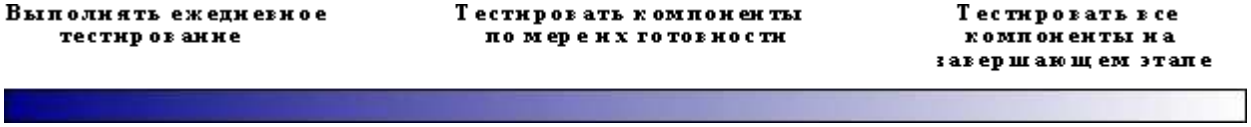


Рис. 1.4. Когда тестировать

Студентам предлагается тестировать компоненты по мере их готовности.

Как надо тестировать?

Основные подходы к тестированию ПО основаны на спецификации и реализации (рис. 1.5).

Спецификация модуля (или класса) ПП определяет, что этот модуль должен делать, т.е. она описывает допустимые наборы входных данных, подаваемых на вход модуля, включая ограничения на то, как многократные вводы данных должны соотноситься друг с другом, и какие выходные данные соответствуют различным наборам входных данных.

Реализация модуля ПП есть выражение алгоритма, порождающего выходные результаты для различных наборов входных данных с соблюдением требований спецификации. Спецификация указывает, что делает модуль ПП, а реализация показывает,

как модуль ПП это делает. Полный учет требований спецификации дает гарантию того, что ПП выполняет все, что от него требуется. Полный учет требований к реализации дает гарантию того, что ПП не будет делать того, что от него не требуется.

Спецификация играет важную роль в тестировании. Обычно для множества компонентов ПП создаются спецификации, обеспечивающие разработку и тестирование, включая спецификации систем, подсистем и классов.

Наряду с автономным тестированием компонентов (классов) системы (модульным уровнем тестирования), необходимо тестировать взаимодействие между различными компонентами (интеграционный уровень тестирования). Цель интеграционного тестирования заключается в обнаружении отказов, возникающих вследствие ошибок в интерфейсах или в силу неверных предположений относительно интерфейсов. После интеграционного тестирования проводится системное тестирование ПП (системный уровень). На этом уровне тестированию подвергается система как единое целое.

Тестирование следует осуществлять в достаточных объемах, чтобы быть более-менее уверенным в том, что ПП функционирует в соответствии с предъявленными к нему требованиями, т.е. выполняется принцип адекватности тестирования ПП. Адекватность можно измерить, используя понятие покрытия. Покрытие можно измерить двумя способами. Первый заключается в подсчете количества требований, сформулированных в спецификации, которые подверглись тестированию. Второй способ заключается в подсчете выполненных компонентов ПП в результате прогона тестового набора. Набор тестов можно считать адекватным, если определенная часть строк исходного кода или исполняемых ветвей исходного кода была выполнена, по крайней мере, один раз во время прогона тестового набора. Эти два способа измерения отражают два базовых подхода к тестированию:

- при использовании первого подхода проверяется, что должен выполнять ПП;
- при использовании второго подхода проверяется, как фактически работает ПП.

При тестировании в соответствии со спецификацией (функциональном тестировании или тестировании "черного ящика") построение тестовых случаев производится в соответствии со спецификацией и не зависит от того, как реализован ПП. Эффективность зависит от качества спецификации и способности тестировщика корректно ее интерпретировать.

При структурном тестировании (тестировании в соответствии с реализацией или тестировании "белого ящика") построение тестовых случаев производится на основе программного кода, представляющего собой реализацию ПП. Входные данные каждого тестового случая должны быть определены спецификацией ПП, однако они могут быть выбраны на основе анализа самого программного кода для прохождения той или иной ветви программы. При этом покрытие увеличивается.

Рис. 1.5. Как тестировать

Практическая работа №5. Планирование тестирования.

Мы будем использовать оба подхода. При тестировании классов мы будем стремиться покрыть как спецификации классов, так и код их реализации. При тестировании взаимодействий будем покрывать спецификацию. При системном тестировании также будем стремиться покрыть спецификацию системы.

В каком объеме тестировать?

Различные уровни адекватного тестирования изображены на рис. 1.6, который охватывает случаи от отсутствия тестирования до исчерпывающего тестирования, когда выполняется прогон всех возможных тестовых случаев. Объем необходимого тестирования следует определять исходя из краткосрочных и долгосрочных целей проекта и в соответствии с особенностями разрабатываемого ПП. Покрытие - это мера полноты использования возможностей программного компонента тестовым набором.

Например, одна из мер - задействована ли каждая строка программного кода продукта хотя бы один раз при прогоне данного тестового набора. Другая мера - количество требований спецификации, проверенных данным тестовым набором. Если требования сформулированы в терминах случаев использования, то покрытие измеряется количеством случаев использования и числом сценариев, построенных для каждого случая использования.

Анализ рисков в процессе тестирования применяется для определения уровня детализации и времени, затрачиваемого на тестирование конкретного компонента. Например, на тестирование классов, более важных для приложения, отводится больше

времени.

Рис. 1.6. В каком объеме тестировать

Мы будем использовать все перечисленные меры.

Ответы на поставленные вопросы и, возможно, на многие другие, оформляются в виде набора документов, принятого в компании. Например, тестовый план может содержать следующую информацию:

1. Перечень тестовых ресурсов.
2. Перечень функций и подсистем, подлежащих тестированию.
3. Тестовую стратегию:

- Анализ функций и подсистем с целью определения слабых мест, требующих исчерпывающего тестирования, то есть участков функциональности, где появление дефектов наиболее вероятно.
- Определение стратегии выбора входных данных для тестирования. Поскольку в реальных применениях множество входных данных программного продукта практически бесконечно, выбор конечного подмножества для проведения тестирования является сложной задачей. Для ее решения могут быть применены методы покрытия классов входных и выходных данных, анализ крайних значений, покрытие случаев использования и тому подобное. Выбранная стратегия должна быть обоснована и задокументирована.
- Определение потребности автоматизации процесса тестирования. При этом решение об использовании существующей, либо о создании новой автоматизированной системы тестирования должно быть обосновано, а также продемонстрирована оценка затрат на создание новой системы или на внедрение уже существующей.

4. График (расписание) тестовых циклов.
5. Указание конкретных параметров аппаратуры и программного окружения.
6. Определение тестовых метрик, которые необходимо собирать и анализировать, таких как покрытие набора требований, покрытие кода, количество и уровень серьезности дефектов, объем тестового кода и т.п.

Тема 3.3 Идентификация и устранение ошибок в информационной системе

Практическая работа №6. Разработка плана резервного копирования

Цель: научиться выполнять архивирование данных и пользоваться службой восстановления системы.

Задание 1. Выполните резервное копирование системных конфигурационных файлов.

1. Запустите виртуальную машину VM-1 и загрузите ОС Windows.
2. Запустите Мастер Архивации **(Пуск/Программы/Стандартные/Служебные/Архивация данных).** Ознакомьтесь с информацией мастера и щелкните **Далее.**

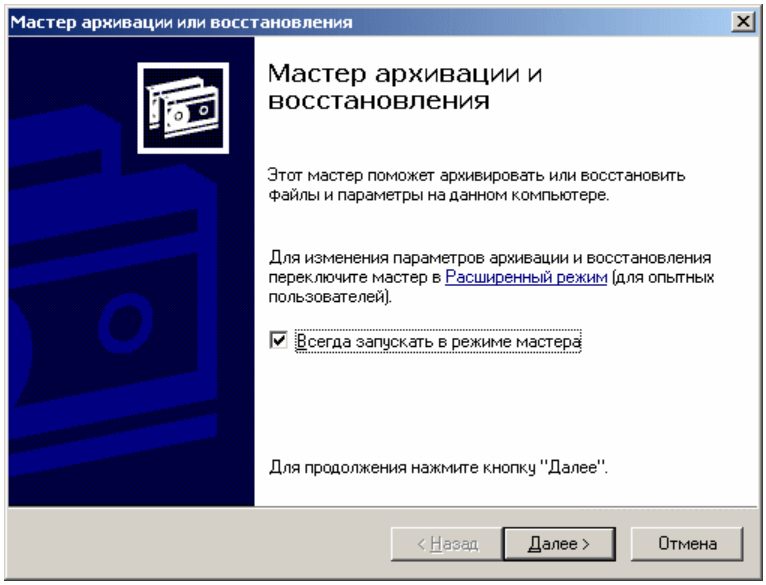


Рисунок 1. Мастер Архивации данных.

3. Выберите возможность мастера – **Архивация файлов** и параметров и щелкните **Далее**
4. Выберите возможность мастера – **Архивация файлов** и параметров и щелкните **Далее.**

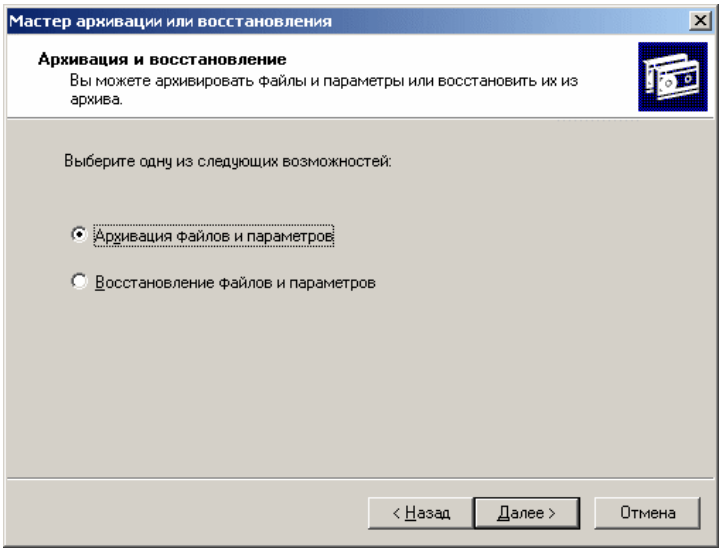


Рисунок 2. Выбор архивации или восстановления.

5. Укажите выбор элементов архивирования в самостоятельном режиме – Предоставить возможность выбора объектов для архивации и щелкните Далее.

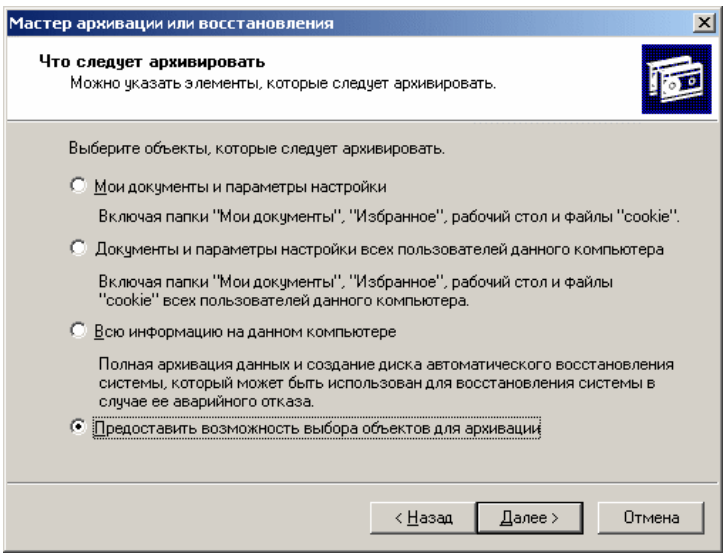


Рисунок 3. Выбор способа указаний объектов архивирования.

6. Укажите элементы для архивации – папки **Documents and Settings** и **Program Files** и щелкните Далее.

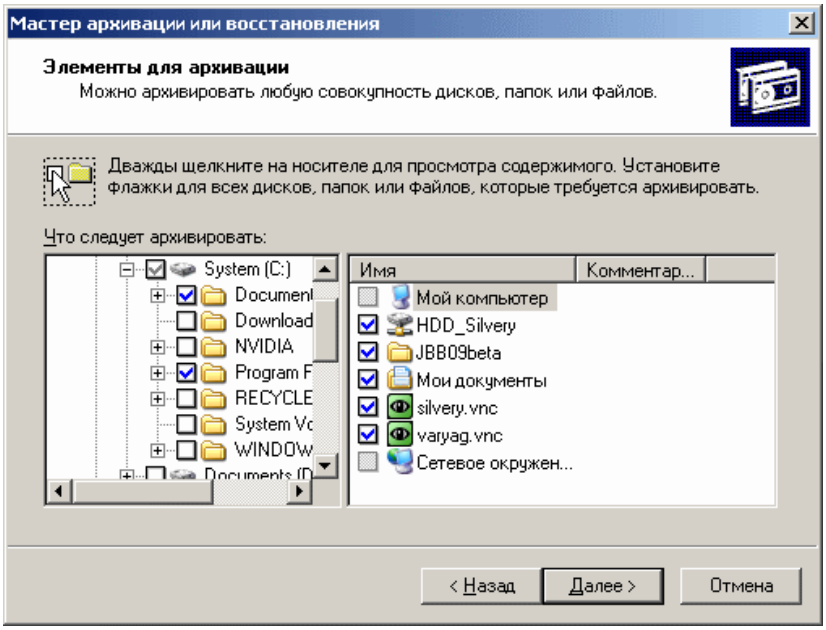


Рисунок 4. Выбор объектов архивирования.

7. Укажите место хранения архива:
8. откройте диалоговое окно Сохранить как кнопкой Обзор;

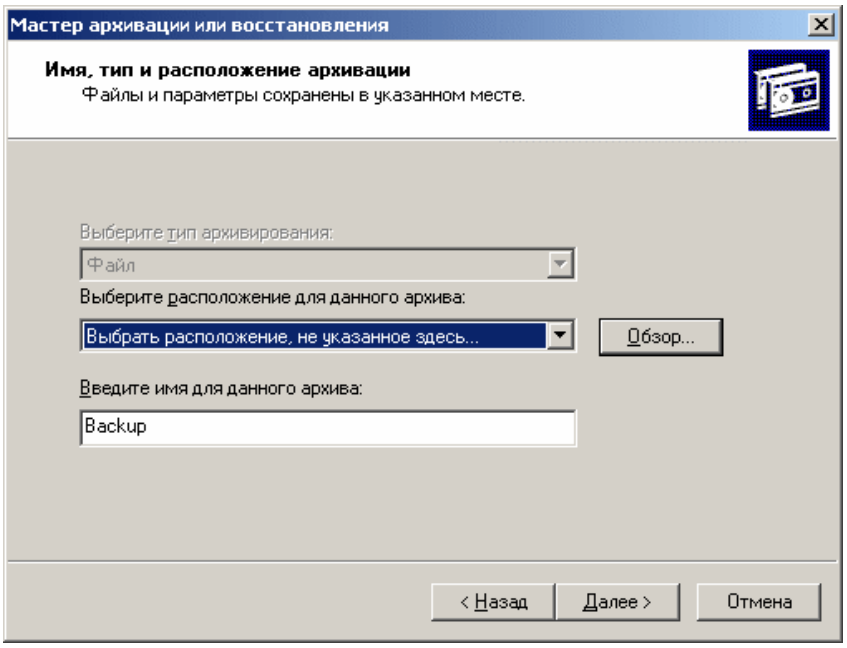


Рисунок 5. Выбор места хранения архива.

- перейдите в корневой каталог диска С;
- введите в поле Имя Файла – имя сохраняемого файла - Резервная Копия;
- сохраните файл кнопкой **Сохранить;**
- подтвердите введенные данные кнопкой **Далее.**

Настройте дополнительные параметры архивации:

- откройте диалоговое окно дополнительных параметров кнопкой ***Дополнительно***;

- выберите в раскрывающемся списке **тип архивации** – *Обычный* и щелкните ***Далее***;
- установите флажок *Проверять данные после архивации* (***Далее***);
- укажите **способ добавления архива** – *Добавить этот архив к существующему* (***Далее***);
- укажите **время архивации**:
    - установите радиокнопку *Позднее*;
    - введите имя задания в соответствующее поле;

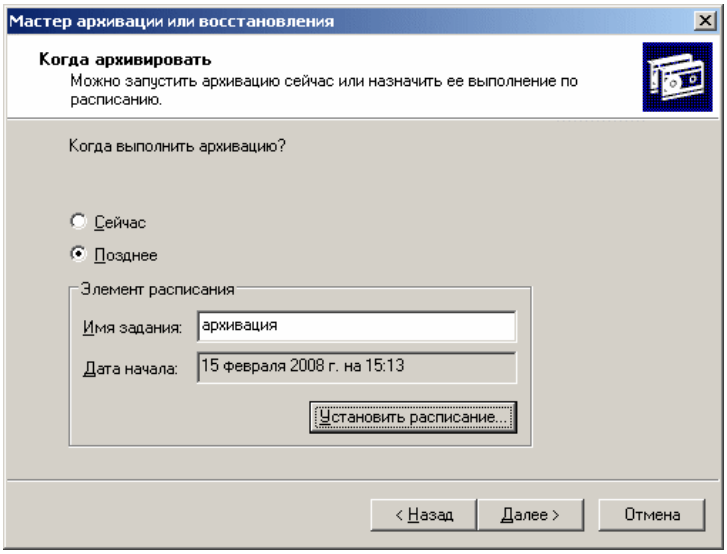


Рисунок 6. Указание имени и времени выполнения архивирования.

- откройте диалоговое окно **Запланированное задание** кнопкой ***Расписание***;

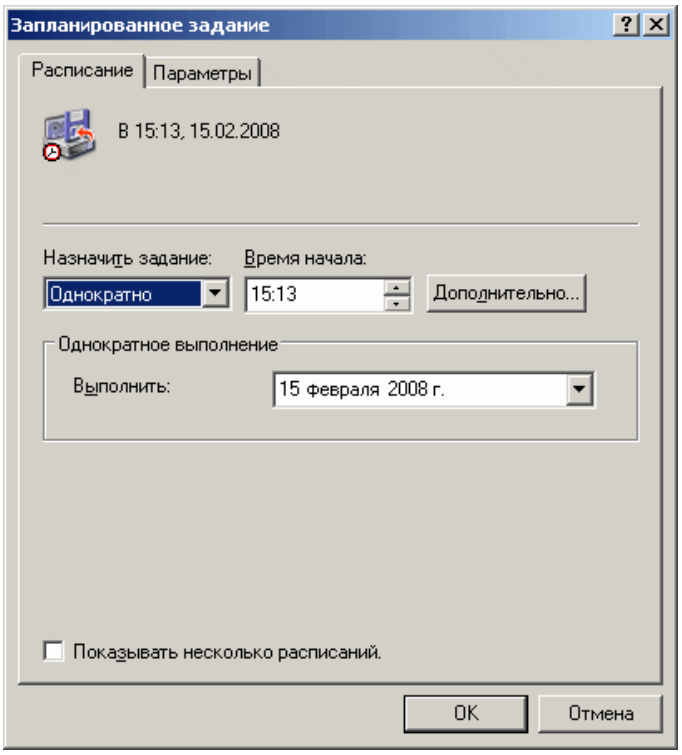


Рисунок 7. Указание точного времени начала выполнения архивирования.

- введите в поле **Время начала** время на *2* минуты позже текущего (например, если сейчас 12.40, то вам необходимо ввести 12.42);

    - подтвердите введенные параметры кнопкой ***ОК***;
- завершите ввод времени выполнения архивации кнопкой ***Далее***;
- введите данные пользователя от имени которого будет выполняться архивирование:
    - введите в поле **Пользователь** имя пользователя на компьютере - *USER*;
    - введите в поля **Пароль** и **Подтверждение пароля** для пользователя *USER*;
    - подтвердите ввод данных кнопкой ***ОК***;

Завершите работу мастера кнопкой ***Готово***.

Задание 2. Выполните восстановление системных конфигурационных файлов.

1. Запустите **Мастер Архивации (*Пуск/Программы/Стандартные/Служебные/Архивация данных*)**.
2. Ознакомьтесь с информацией мастера и щелкните ***Далее***.
3. Выберите возможность мастера – **Восстановление файлов и параметров** и щелкните ***Далее***.
4. Выберите для восстановления в левом списке с содержимым архива, папку ***Мои рисунки*** (***Далее***);.
5. Ознакомьтесь с выбранными параметрами и активизируйте восстановление кнопкой ***Готово***.
6. Откройте отчет кнопкой ***Отчет*** и просмотрите его.
7. Закройте диалоговое окно **Ход восстановления** кнопкой ***Закрыть***.

Задание 3. Создайте точку восстановления.

1. Запустите мастер **Восстановление системы (*Пуск/Программы/Стандартные/Служебные*)**.
2. Ознакомьтесь с информацией мастера.
3. Создайте точку восстановления:

- Установите радиокнопку *Создать точку восстановления* (***Далее***);
- введите в текстовое поле **Описание контрольной точки восстановления** - *Тестовая точка восстановления*;

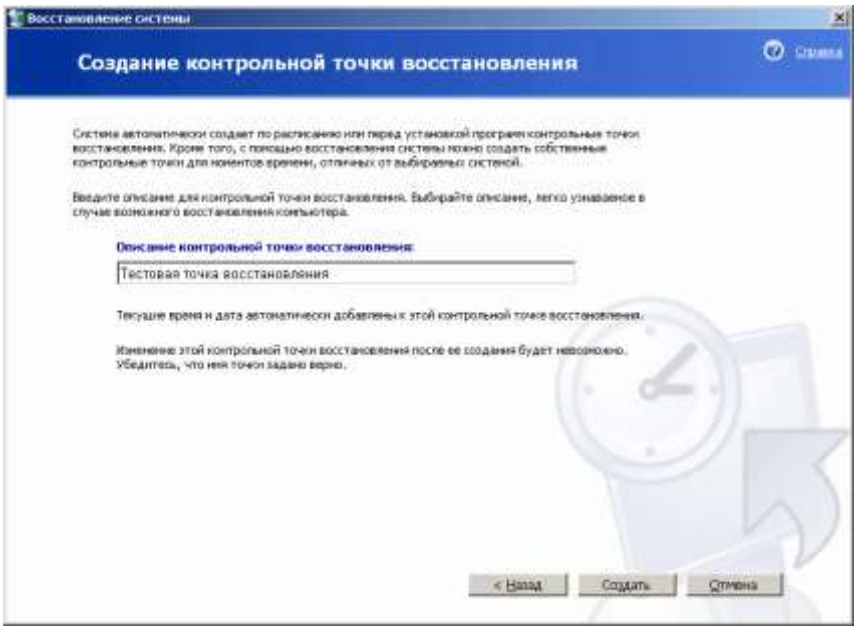


Рисунок 8. Восстановление системы.

- создайте точку восстановления кнопкой ***Создать***.

Завершите работу мастера кнопкой ***Закрыть***.

Практическая работа №7. Сбор информации об ошибках. Формирование отчетов об ошибках.

Цель: научится собирать информацию об ошибках. Формировать отчеты об ошибках.

Отчёт об ошибке (англ. error report или crash report) — это файл, содержащий техническую информацию об исключительной ситуации (исключении), произошедшей в программе на компьютере пользователя.

Отчеты об ошибках создаются, когда в системе возникают неполадки с аппаратным или программным обеспечением. Эти отчеты активно используются корпорацией Майкрософт и ее партнерами для повышения надежности программного обеспечения.

Отчеты об ошибках содержат следующие разделы: сведения о состоянии сервере при возникновении ошибки, версия используемой операционной системы и аппаратного обеспечения, а также цифровой код продукта, используемый для определения лицензии. Также передается IP-адрес компьютера, поскольку для отправки отчета необходимо подключиться к сетевой службе, но он используется только для сбора статистических данных. Корпорация Майкрософт не собирает специально персональные данные.

Однако отчеты об ошибках могут содержать данные файлов журналов, например, имена пользователей, IP-адреса, URL-адреса, имена файлов и пути к ним, а также адреса электронной почты. Однако эти сведения (если они присутствуют) не используются для установления личности пользователя, хотя это теоретически возможно. Собранные корпорацией Майкрософт данные используются только для устранения неполадок в работе системы и улучшения работы программ и служб. Отчеты об ошибках отправляются с использованием шифрования в базу данных с ограниченным доступом и не используются в каких-либо коммерческих целях.

Отчет об ошибках в Windows XP Отчет ошибках может предоставить всю информацию, которая требуется для решения возникающих проблем, и особенно проблем, которые приводят к появлению стоп-ошибок (так же известных, как "синие экраны смерти").

Система аварийно завершает свою работу, перезагружается и выясняет у пользователя необходимость отправки сообщения об ошибке компании Microsoft. Достаточно щелкнуть на кнопке Отправить отчет об ошибке (Send Error Report) и это можно исправить.

Операционная система буквально собирает информацию об аварийном завершении работы, анализирует ее и сообщает пользователю о подсистеме, которая нуждается в исправлении. Конечно, для использования этой возможности необходимо ее включить. Хотя эта система включена по умолчанию, стоит убедиться, что служба Error Reporting включена и активно ищет проблемы, о которых необходимо сообщить производителю.

Для проверки включения службы Error Reporting выполните такую последовательность действий:

1. Кликните на кнопке Пуск (Start) и щелкните правой кнопкой на ссылке Мой компьютер (My Computer). Выберите в контекстном меню команду Свойства (Properties).

2. Перейдите на вкладку Дополнительно (Advanced) и кликните на кнопке Отчет об ошибках.

Структура отчета об ошибке ID (Идентификатор, Номер). Каждой записи в системе

учета ошибок присваивается уникальный идентификатор или номер. Как правило, он задается самой системой по определенному шаблону. Это может быть просто числовой номер (1,2,3,…3487), а может быть идентификатор вида ПРОЕКТ-НОМЕР, например, TPO-2367, REG-335 и так далее. Тип (Трекер). Системы управления задачами, как правило, содержат в себе записи различных типов - задача (Task), улучшение (Feature), баг (Bug), пользовательская история (User Story) и так далее. Для каждого типа может быть свой набор полей и свой жизненный цикл. Заголовок (Тема, Title). Краткое описание проблемы. Оно отображается в списках, результатах поиска, фильтрах и позволяет быстро понять, в чем суть проблемы. Оно не должно быть слишком коротким и общим, но одновременно и не должно быть слишком длинным.

Существует мнемоническое правило для грамотного составления описания «Что? Где? Когда?»: описание должно описывать, что именно сломалось, где сломалось и при каких условиях. Например, «Не работает поиск» — плохое описание ошибки, а «В форме поиска после отправки запроса выдается ошибка „Internal Error“ вместо результатов» — уже лучше.

Проект. Как правило, один большой продукт подразделяется на несколько проектов для более удобного управления, и в системе учета задач необходимо указать, к какому именно проекту относится данная ошибка. Версия. Версия продукта, в которой ошибка была обнаружена. Компонент.

Компонент продукта, где была обнаружена ошибка. Как правило, список доступных компонентов ограничен и создается администратором системы.

Приоритет (Priority). Приоритет показывает, с какой срочностью должен быть исправлен дефект.

Серьезность (Важность, Severity). Серьезность показывает степень влияния дефекта на систему.

Окружение. Описание системы - программного и аппаратного обеспечения, на котором воспроизводится данный дефект. На кого назначен. Кто будет ответственен за решение данного дефекта.

В зависимости от принятых правил и процессов в компании, это может быть конкретный разработчик, руководитель группы разработчиков, или же поле по умолчанию остается пустым, а разработчики сами решают, кто будет править этот дефект Шаги для воспроизведения. Не всегда этот пункт выделяется как отдельное поле. Если его нет, то нужно шаги для воспроизведения написать в поле «Описание».

Фактический результат. Должен быть обязательно - нужно указать, что мы получили в результате выполнения указанных шагов.

Ожидаемый результат. Необходимо указать, чтобы было понятно, как система должна работать. Если есть возможность, то необходимо давать ссылку на документацию, требования или иные документы, в которых описывается ожидаемое поведение системы. В некоторых случаях этот пункт можно опустить, если ожидаемый результат тривиален (сервер отдает ошибку 500, программа аварийно завершается и т.п.)

Вложения (скриншоты, логи и пр.). Файлы, которые могут помочь для понимания, воспроизведения, локализации и исправления дефекта.

Выполнение работы

1. Изучить тему «Отчет об ошибках системы».

2. Укажите порядок подключения службы Error Reporting.

3. Укажите порядок настройки Отчетов об ошибках. Какие виды ошибок включены в

Отчет об ошибках.

4. Изучите структуру отчета об ошибках и приведите пример записи ошибки.

Практическая работа №8. Особенности применения методик стохастического тестирования и метод оценки скорости выявления ошибок. Выбор тестовых случаев.

Исчерпывающее тестирование, другими словами, прогон каждого возможного тестового случая, покрывающего каждое сочетание значений - это, вне всяких сомнений, надежный подход к тестированию. Однако во многих ситуациях количество тестовых случаев достигает таких больших значений, что обычными методами с ними справиться попросту невозможно. Если имеется принципиальная возможность построения такого большого количества тестовых случаев, на построение и выполнение которых не хватит никакого времени, должен быть разработан систематический метод определения, какими из тестовых случаев следует воспользоваться. Если есть выбор, то мы отдаем предпочтение таким тестовым случаям, которые позволяют найти ошибки, в обнаружении которых мы заинтересованы больше всего.

Существуют различные способы определения, какое подмножество из множества всех возможных тестовых случаев следует выбирать. При любом подходе мы заинтересованы в том, чтобы систематически повышать уровень покрытия.

Подробное описание тестового случая

Продемонстрируем тестирование взаимодействий на примере класса TCommandQueue. Из табл. 3.2, которая была составлена на основе спецификаций классов, описанных в приложении 2, видно, что класс очереди команд взаимодействует со следующими классами: TBearingParam,

TAxleParam,

TCommand,

TStore,

TterminalBeari

ng.

С объектом TCommand осуществляется взаимодействие третьего типа, т. е. TCommandQueue создает объекты класса TCommand как часть своей внутренней реализации. С остальными классами осуществляется взаимодействие первого типа: ссылки на объекты
классов TStore и TTerminalBearing передаются как параметры в конструктор TCommandQueue, а ссылки на объекты классов TBearingParam и TAxleParam передаются в метод TCommandQueue.AddCommand.

Одновременно с изучением этого раздела можно открыть проект IntegrationTesting\IntegrationTests.sln.

Для тестирования взаимодействия класса TCommandQueue и класса TCommand, так же, как и при модульном тестировании, разработаем спецификацию тестового случая:

Таблица 3.3. Спецификация тестового случая

Названия взаимодействующих классов: Название теста: TCommandQueue, TCommand TCommandQueueTest1

Описание теста: тест проверяет возможность создания объекта типа TCommand и добавления его в очередь при вызове метода AddCommand Начальные условия: очередь команд пуста

Ожидаемый результат: в очередь будет добавлена одна команда

На основе этой спецификации был разработан тестовый драйвер - класс TCommandQueueTester, который наследуется от класса Tester. Этот класс содержит:

- Метод Init, в котором создаются объекты классов TStore, TterminalBearing и объект типа TcommandQueue. Этот метод необходимо вызывать в начале каждого теста, чтобы тестируемые объекты создавались вновь:
- private void Init()
- {
- TB = new TTerminalBearing();
- S = new TStore();
- CommandQueue=new TCommandQueue(S,TB);
- S.CommandQueue=CommandQueue;

}

Пример 3.1. Метод Init

- Методы, реализующие тесты. Каждый тест реализован в отдельном методе.
- Метод Run, в котором вызываются методы тестов.
- Метод dump, который сохраняет в log-файле теста информацию обо всех командах, находящихся в очереди в формате - номер позиции в очереди: полное название команды.
- Точку входа в программу - метод Main, в котором происходит создание экземпляра класса TCommandQueueTester и запуск метода Run.

Сначала создадим тест, который проверяет, создается ли объект типа TCommand, и добавляется ли команда в конец очереди.

```
private void TCommandQueueTest1()
{
Init();
LogMessage("/////////// TCommandQueue Test1 ////////////");
LogMessage("Проверяем, создается ли объект типа TCommand");
// В очереди нет команд dump();
// Добавляем команду
// параметр = -1 означает, что команда должна быть добавлена
//в конец очереди
CommandQueue.AddCommand(TCommand.GetR,0,0,0,n
ew TBearingParam(),new TAxleParam(),-1);
LogMessage("Command added");
// В очереди одна команда
```

dump(); }

Пример 3.2. Тест, проверяющий создание объекта типа TCommand В этот класс включены еще два разработанных теста.

Тема. Организация сопровождения и восстановления работоспособности системы.

Практическая работа №9 Интеграционное тестирование. Идентификация взаимодействий.

Основное назначение тестирования взаимодействий состоит в том, чтобы убедиться, что происходит правильный обмен сообщениями между объектами, классы которых уже прошли тестирование в автономном режиме (на модульном уровне тестирования).

Тестирование взаимодействия или интеграционное тестирование представляет собой тестирование собранных вместе, взаимодействующих модулей (объектов). В интеграционном тестировании можно объединять разное количество объектов - от двух до всех объектов тестируемой системы. Интеграционное тестирование отличается от системного тем, что:

- при интеграционном тестировании используется подход "белого ящика", а при системном - "черного ящика";
- целью интеграционного тестирования является только проверка правильности взаимодействия объектов, тогда как целью системного - проверка правильности функционирования системы в целом.

Идентификация взаимодействий

Взаимодействие объектов представляет собой просто запрос одного объекта (отправителя) на выполнение другим объектом (получателем) одной из операций получателя и всех видов обработки, необходимых для завершения этого запроса.

В ситуациях, когда в качестве основы тестирования взаимодействий объектов выбраны только спецификации общедоступных операций, тестирование намного проще, чем когда такой основой служит реализация. Мы ограничимся тестированием общедоступного интерфейса. Такой подход вполне оправдан, поскольку мы полагаем, что классы уже успешно прошли модульное тестирование. Тем не менее, выбор такого подхода отнюдь не означает, что не нужно возвращаться к спецификациям классов, дабы убедиться в том, что тот или иной метод выполнил все необходимые вычисления. Это обусловливает необходимость проверки значений атрибутов внутреннего состояния получателя, в том числе любых агрегированных атрибутов, т.е. атрибутов, которые сами являются объектами. Основное внимание уделяется отбору тестов на основе спецификации каждой операции из общедоступного интерфейса класса.

Взаимодействия неявно предполагаются в спецификации класса, в которой установлены ссылки на другие объекты. В разделе 4 рассматривалось тестирование примитивных классов. Такие объекты представляют собой простейшие компоненты системы и, несомненно, играют важную роль при выполнении любой программы. Тем не менее, в объектно-ориентированной программе существует сравнительно небольшое количество примитивных классов, которые реалистично моделируют объекты задачи и все отношения между этими объектами. Обычным явлением для хорошо спроектированных объектно-ориентированных программ является использование непримитивных классов; в этих

программах им отводится главенствующая роль.

Выявить такие взаимодействующие классы можно, используя отношения ассоциации (в том числе отношения агрегирования и композиции), представленные на диаграмме классов. Ассоциации такого рода преобразуются в интерфейсы класса, а тот или иной класс взаимодействует с другими классами посредством одного или нескольких способов:

Тип 1. Общедоступная операция имеет один или большее число формальных параметров объектного типа. Сообщение устанавливает ассоциацию между получателем и параметром, которая позволяет получателю взаимодействовать с этим параметрическим объектом.

Тип 2. Общедоступная операция возвращает значения объектного типа. На класс может быть возложена задача создания возвращаемого объекта, либо он может возвращать модифицированный параметр.

Тип 3. Метод одного класса создает экземпляр другого класса как часть своей реализации.

Тип 4. Метод одного класса ссылается на глобальный экземпляр некоторого другого класса. Разумеется, принципы хорошего тона в проектировании рекомендуют минимальное использование глобальных объектов. Если реализация какого-либо класса ссылается на некоторый глобальный объект, рассматривайте его как неявный параметр в методах, которые на него ссылаются.

Приведем еще раз таблицу разделения классов на примитивные и непримитивные типы (табл. 3.1):

Таблица 3.1. Типы классов

Класс	**Тип**

TBearingParam	Примитивный
TAxleParam	Примитивный
TCommand	Примитивный
TLog	Примитивный
TCommandQueue	Непримитивный
Класс	Непримитивный
TStore	Непримитивный
TTerminalBearing	Непримитивный
TTerminalAxle	Непримитивный
TModel	Непримитивный
MainForm	Непримитивный

Таблица 3.2. Типы взаимодействия классов

Непримитивные типы	Tbearing Param	Taxle Param	Tcommand	TCommand Queue	Store	TTerminal Bearing	TTerminal Axle	Log	model
TCommandQueue	1	1	3			1			
TStore	3	1	1						
TTerminalBearing	3								
TTerminalAxle		3							
TModel				3		3	3		
MainForm									

Таким образом, интеграционному тестированию будут подвергнуты взаимодействия перечисленных непримитивных классов.

Практическая работа №10. Интеграционное тестирование. Описание тестовых процедур.

Описание тестовых процедур Как запустить тест?

Для выполнения этого теста в методе Run необходимо вызвать метод TCommandQueueTest1() и запустить программу на выполнение:

```
private void Run()
{
  TCommandQueueTest1();
}
```

Пример 3.3. Метод Run

Проверка результатов выполнения тестов (сравнение с ожидаемым результатом) После завершения теста следует просмотреть текстовый журнал теста (..\IntegrationTesting\bin\ Debug\test.log), чтобы сравнить полученные результаты с ожидаемыми результатами, заданными в спецификации тестового случая TCommandQueueTest1.

```
////////////////////  TCommandQueue  Test1
//////////////////  Проверяем,  создается  ли
объект  типа  TCommand  0  commands  in
command queue
Command added
1 commands in command queue
0 : ПОЛУЧИТЬ ИЗ ВХОДНОЙ ЯЧЕЙКИ
```

Пример 3.4. Журнал теста

Задание 2

Для тестирования взаимодействия остальных непримитивных классов (табл. 2.1) по аналогии с приведенным примером требуется самостоятельно разработать спецификации тестовых случаев, соответствующие тесты, провести тестирование и составить тестовые отчеты (табл. 2.2).

Практическая работа №11. Выполнение тестирования.

Задание. Для тестового случая №1 необходимо составить полный список всех возможных альтернативных путей (см. подраздел "Список альтернативных путей") и разработать соответствующие тесты.

Кроме того, необходимо:

- выбрать случай использования на основании дерева решений (..\SystemTesting\Decision Tree.vsd);
- составить пошаговое описание выбранного случая использования;
- учесть все альтернативные пути;
- составить спецификации тестовых случаев;
- разработать соответствующие тестовые случаи (тесты) ; с составить тестовые отчеты (табл. 2.2).

Практическая работа №12. Системное тестирование

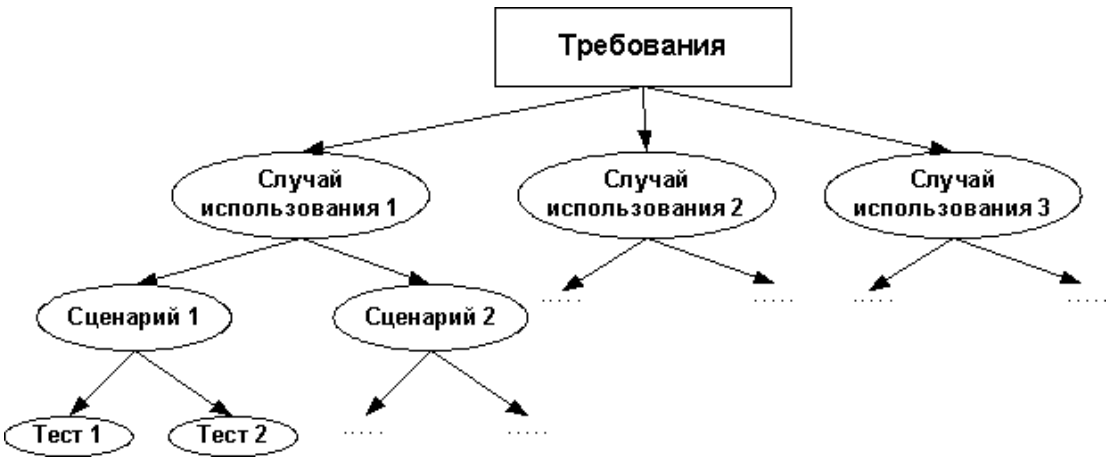
Системное тестирование качественно отличается от интеграционного и модульного уровней. Системное тестирование рассматривает систему в целом и применяется на уровне пользовательских интерфейсов, в отличие от последних фаз интеграционного тестирования, которое оперирует на уровне интерфейсов модулей, хотя набор модулей может быть аналогичным. Различны и цели этих уровней тестирования. На уровне системы часто сложно и малоэффективно анализировать прохождение тестовых траекторий внутри программы, а также отслеживать правильность работы конкретных функций. Основной задачей системного тестирования является выявление дефектов, связанных с работой системы в целом, таких как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство в использовании и тому подобное.

Поскольку системное тестирование проводится на уровне пользовательских интерфейсов, то построение специальной тестовой системы становится технически необязательным. Однако объемы данных на этом уровне таковы, что обычно более эффективным подходом является полная или частичная автоматизация тестирования, что может потребовать создания тестовой системы намного более сложной, чем система тестирования на уровне модулей или их комбинаций.

Необходимо подчеркнуть, что существует два принципиально разных подхода к системному тестированию.

В первом варианте для построения тестов используются требования к системе, например, для каждого требования строится тест, который проверяет выполнение данного требования в системе. Этот подход особенно широко применяется при разработке военных и научных систем, когда заказчик вполне осознает, какая функциональность ему нужна, и составляет полный набор формальных требований. Тестировщик в данном случае только проверяет, соответствует ли разработанная система этому набору. Такой подход предполагает длинную и дорогостоящую фазу сбора требований, выполняемую до начала собственно проекта. В этом случае для определения требований обычно разрабатывается прототип будущей системы.

Во втором подходе основой для построения тестов служит представление о способах использования продукта и о задачах, которые он решает. На основе более или менее формальной модели пользователя создаются случаи использования системы, по



которым затем строятся собственно тестовые случаи. Случай использования (use case) описывает, как субъект использует систему, чтобы выполнить ту или иную задачу. Субъекты или актеры (actors) могут исполнять различные роли при работе с системой. Случаи использования могут описываться с различной степенью абстракции. Случаи использования не обязательно охватывают каждое требование. Можно конкретизировать случаи использования и расширять их в наборы более специфических случаев использования (пошаговое описание случая использования). В контексте конкретного случая использования можно определить один или большее число сценариев. Сценарий представляет конкретный экземпляр случая использования - путь в пошаговом описании случая использования. Каждый путь (сценарий) в случае использования должен быть протестирован (рис. 4.1).

Рис. 4.1. Тестирование случаев использования

Входные данные для каждого сценария надо выбирать следующим образом:

- Идентифицировать все значения (входные данные), которые могут задавать субъекты для случая использования.
- Определить классы эквивалентности для каждого типа входных данных.
- Построить таблицу со списком значений из различных классов эквивалентности.
- Построить тестовые случаи на базе таблицы с учетом внешних ограничений.

Далее при построении тестовых случаев применялись оба подхода и при выполнении заданий необходимо действовать следующим образом:

- На основе требований определить случаи использования (use case)
- На основе каждого случая использования (use case) построить сценарии.
  с Для каждого сценария разработать тестовые случаи (набор тестов).

Практическая работа №13. Модульное тестирование на примере классов. Тестовый прогон.

Цель тестирования программных модулей состоит в том, чтобы удостовериться, что каждый модуль соответствует своей спецификации. Если это так, то причиной любых ошибок, которые возникают при их объединении, является неправильная стыковка модулей. В процедурно- ориентированном программировании модулем называется процедура или функция, иногда группа процедур, которая реализует абстрактный тип данных. Тестирование модулей обычно представляет собой некоторое сочетание проверок и прогонов тестовых случаев. Можно составить план тестирования модуля, в котором учесть тестовые случаи и построение тестового драйвера.

Тестирование классов аналогично тестированию модулей. Основным элементом объектно-ориентированной программы является класс. Рассмотрим методику тестирования отдельного класса. Тестирование классов охватывает виды деятельности, ассоциированные с проверкой реализации класса на точное соответствие спецификации класса. Если реализация корректна, то каждый экземпляр этого класса ведет себя подобающим образом.

Эффективного тестирования классов можно достичь при помощи ревью и тестовых прогонов. Ревью представляет собой просмотр исходного кода ПО с целью обнаружения

ошибок и дефектов, возможно, до того, как это ПО заработает. Ревьюирование предназначено для выявления таких ошибок, как неспособность выполнять то или иное требование спецификации или ее неправильное понимание, а также алгоритмических ошибок в реализации. Тестовый прогон обеспечивает тестирование ПО в процессе выполнения программы. Осуществляя прогон программы, тестировщик стремится определить, способна ли программа вести себя в соответствии со спецификацией. Тестировщик должен выбрать наборы входных данных, определить соответствующие им правильные наборы выходных данных и сопоставить их с реально получаемыми выходными данными.

Рассмотрим тестирование классов в режиме прогона тестовых случаев. После идентификации тестовых случаев для класса нужно реализовать тестовый драйвер, обеспечивающий прогон каждого тестового случая, и запротоколировать результаты каждого прогона. При тестировании классов тестовый драйвер создает один или большее число экземпляров тестируемого класса и осуществляет прогон тестовых случаев. Тестовый драйвер может быть реализован как автономный тестирующий класс.

Кто, что, когда, как и в каком объеме?

Рассмотрим эти вопросы в контексте тестирования классов.

Кто выполняет тестирование? Обычно тестирование классов выполняют их разработчики. В этом случае время на изучение спецификации и реализации сводится к минимуму. Недостатком подхода является то, что если разработчик неправильно понял спецификации, то он для своей неправильной реализации разработает и "ошибочные" тестовые наборы.

Что тестировать? Необходимо удостовериться, что программный код класса в точности отвечает требованиям, сформулированным в его спецификации, и что он не делает ничего более.

В какой момент следует выполнять тестирование? План тестирования или хотя бы тестовые случаи должны разрабатываться после составления полной спецификации класса. Разработка тестовых случаев по мере реализации класса помогает разработчику лучше понять спецификацию. Тестирование класса должно проводиться до того, как возникнет необходимость использовать этот класс в других компонентах ПО. Регрессионное тестирование класса должно выполняться всякий раз, когда меняется реализация класса. Регрессионное тестирование позволяет убедиться в том, что разработанные и оттестированные функции продолжают удовлетворять спецификации после выполнения модификации ПО.

Как будет выполняться тестирование? Тестирование классов обычно выполняется путем разработки тестового драйвера, который создает экземпляры классов и окружает эти экземпляры соответствующей средой (тестовым окружением), чтобы стал возможен прогон соответствующего тестового случая. Драйвер посылает сообщения экземпляру класса в соответствии со спецификацией тестового случая, а затем проверяет исход этих сообщений. Тестовый драйвер должен удалять созданные им экземпляры тестируемого класса. Статические элементы данных класса также необходимо тестировать.

Какие объемы тестирования следует считать адекватными? Адекватность может быть измерена полнотой охвата тестами спецификации или реализации. Будем использовать оба способа.

Тема. Оценки сложности тестирования и методика тестирования объектно-ориентированной программы

Практическая работа №14. Описание ручного тестирования.

Реализация выбранного подхода для ручного тестирования приводится в Class1.cs (..\ SystemTesting\ManualTests\Tests\Class1.cs).

Все классы входят в пространство имен Tests. Это пространство имен содержит следующие классы:

- Class1 - главный класс приложения. Содержит статический метод Main, вызываемый при запуске;
- Test - абстрактный (abstract) класс, реализующий общую для всех тестов функциональность. Содержит следующие методы:
  - public Test() - конструктор. Создает серверный сокет и запускает сервер;
  - protected void wait(string st) - ожидает получения от dll вызова, начинающегося со строки st ;
  - protected void finish() - обрабатывает последний запрос от dll и закрывает серверный сокет; ○ virtual public void start() - запускает тест. В каждом конкретном тесте переопределяется.

Кроме того, класс Test содержит пять protected полей типа string:

- StoreStat - статус склада;
- AxlePar - терминал оси;
- RollerPar - терминал подшипника;
- CommandStatus - возвращаемое значение функции SendStoreCom ; с StoreMessage - сообщение от склада.

Как создать свой тест?

Для создания нового теста необходимо выполнить следующие действия:

- создать новый класс, являющийся потомком Test ;
- переопределить в нем метод start(), чтобы реализовать функциональность теста:
  - задать состояние окружения (StoreStat, AxlePar, RollerPar, StoreMessage, CommandStatus); ○ ждать, когда произойдет определенное событие (вызов wait, в котором надо задать строку для выхода из состояния ожидания); ○ задать новое состояние окружения и т.д.

с тест должен завершаться вызовом finish(). В общем виде тест выглядит так:

```
override public void start()
{ <задание переменных> wait(<строка>);
<задание
переменных>
```

```
wait(<строка>);
....
finish();
}
```

Практическая работа №15. Автоматизация тестирования с помощью скриптов. Создание своих тестов.

Как создать свой тест?

В данном случае используется тот же принцип, что и при ручном тестировании:

- задать состояние окружения (StoreStat, AxlePar, RollerPar, StoreMessage, CommandStatus);
- ждать, когда в системе произойдет определенное событие;
- задать новое состояние окружения; с и т.д.

Каждый тест должен быть представлен в следующем

виде: ЗАГОЛОВОК

БЛОК

Wait

<условие>

БЛОК

Wait <условие>

.....

EndTest

Опишем эти элементы более подробно.

Описание заголовка

Заголовок теста состоит из:

- команды запуска сервера;
- команд global, устанавливающих доступ к глобальным переменным состояния окружения;
- команды StartTest, задающей имя теста (оно не обязательно должно совпадать с именем файла).

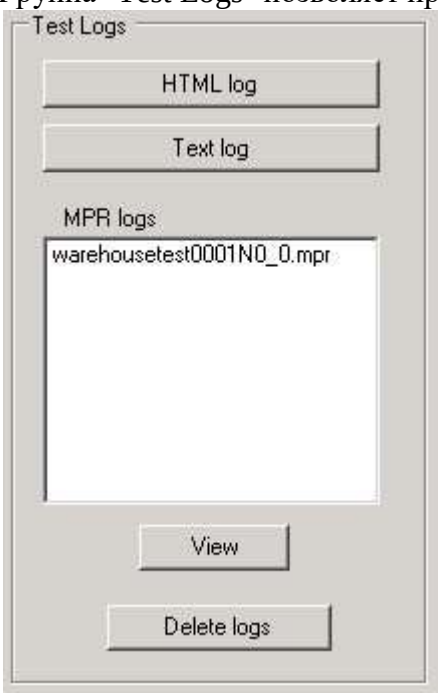
```
source bin\\srv.tcl  // запуск сервера
global StoreStat          // статус склада
global AxlePar
// терминал оси global
RollerPar                              // терминал подшипника
global CommandStatus  // возвращаемое значение функции
                  // SendStoreCom
global StoreMessage   // сообщение от склада
global rollers_found  // 1 (можно подобрать подходящий
                  // подшипник) или 0 (нельзя)
global fds              // строка для графы "Покрытие FDS"
// в итоговой таблице результатов
```

// тестирование (по умолчанию - Default) global
last_command // последняя команда тестируемой системы
global allowed // список разрешенных команд, их
// получение должно вызывать
ошибку StartTest <имя> // запуск теста
Описание блока
Каждый блок устанавливает значения одной или более переменных окружения.
Например:
Set StoreStat 32

Тема 3.6 Автоматизация тестирования структуры тестового набора для автоматического прогона.

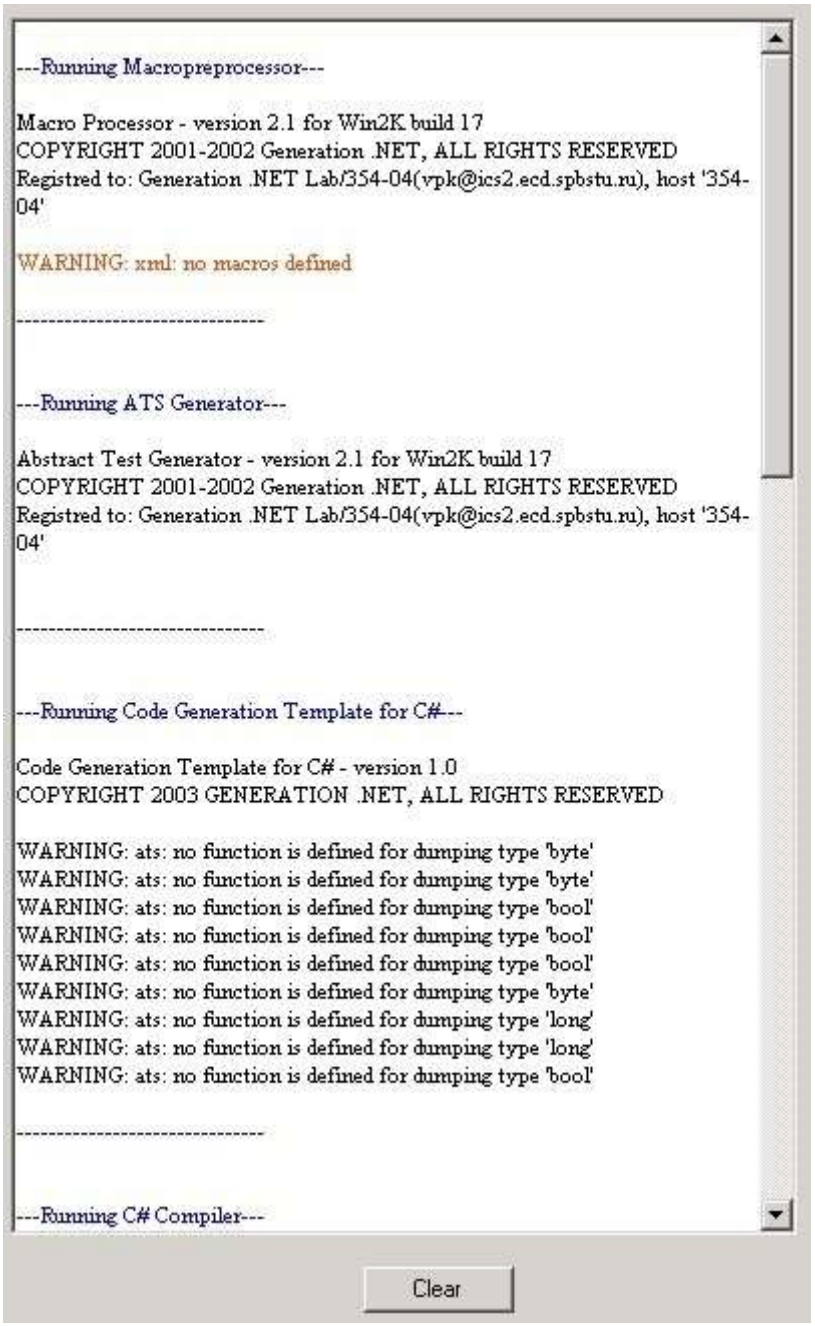
Практическая работа №16. Описание функционала.

Группа "Test Logs" позволяет просматривать протоколы тестирования:

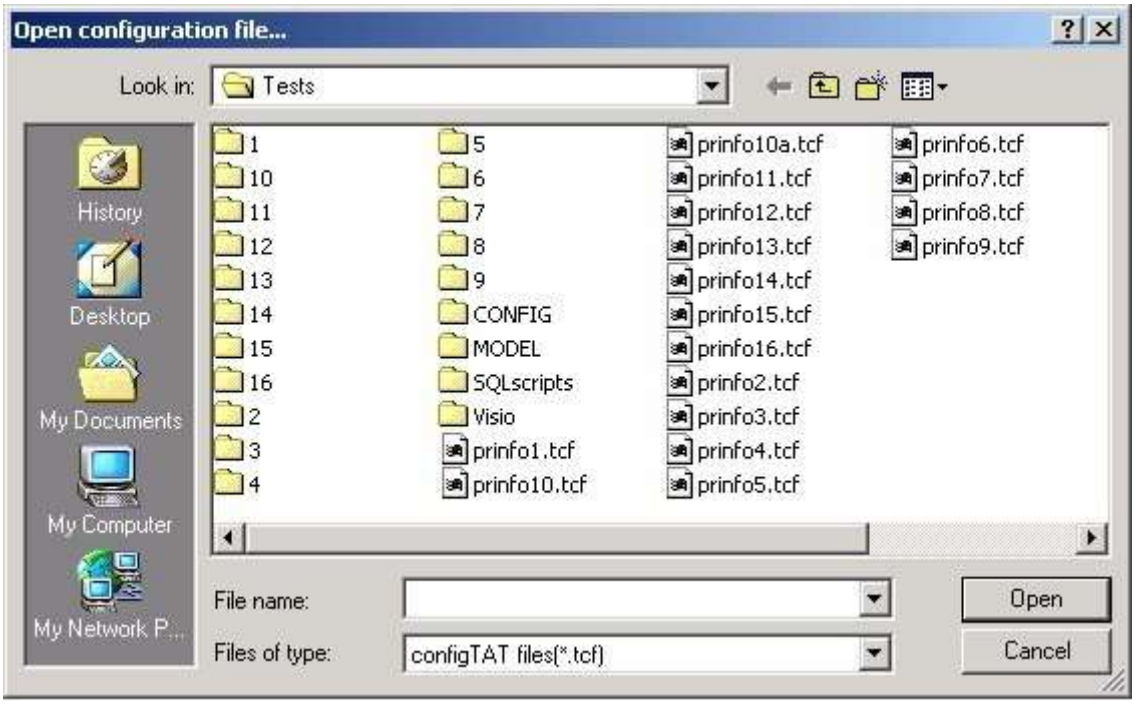


- при нажатии кнопки "HTML log" в Internet Explorer отображается протокол тестирования в виде html-страницы;
- при нажатии кнопки "Text log" в notepad отображается протокол тестирования в виде txt-файла;
- в listBox-е "MPR logs" отображается список протоколов в формате mpr (отдельный протокол для каждого testcase-а и каждой итерации теста), которые можно открыть в программе Telelogic нажатием кнопки "View";
- с помощью кнопки "Delete logs" можно удалить все протоколы тестирования.

В richTextBox-е в правой части формы отображается информация о процессе генерации и выполнении тестов. Очистить richTextBox можно с помощью кнопки "Clear":



Конфигурации тестов можно сохранять и открывать с помощью команд, соответственно, Save и Open меню File:

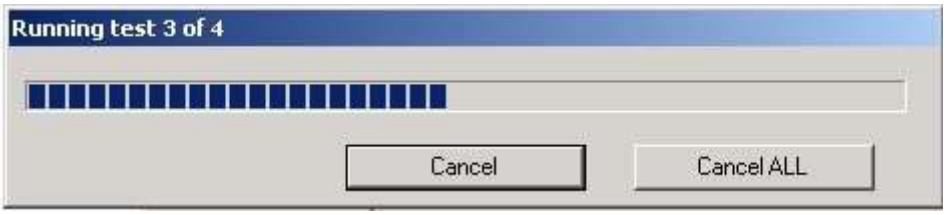


При активной вкладке "TestSuite" осуществляется настройка и запуск набора тестов.

Кнопки "Add" и "Remove" позволяют добавлять и убирать отдельные тесты (файлы конфигурации, созданные на вкладке "Test") из набора тестов.

Кнопка "Run" запускает на выполнение последовательность тестов. При этом все кнопки и меню блокируются и отображается Progress Bar.

Остановить выполнение данного теста или всей тестовой последовательности можно с помощью кнопок "Cancel" и "Cancell ALL" соответственно.



В richTextBox-е в правой части формы отображается краткая информация о результатах тестирования:

С помощью пунктов "Open" и "Save" меню "File" можно, соответственно, загрузить и сохранить список тестов тестового набора.

SysLog Animator Manual

Эта программа предназначена для визуализации журнала системы, полученного в результате тестирования системы).

Журнал системы представляет собой набор строчек следующего вида:

08.09.2003 15:01:29 : СИСТЕМА: Запрошен статус СКЛАДА

08.09.2003 15:01:29 : СКЛАД: Статус СКЛАДА = 16

08.09.2003 15:01:29 : СИСТЕМА: Запрошены параметры оси 08.09.2003 15:01:30 : СИСТЕМА: СКЛАДУ послана команда ПОЛОЖИТЬ В РЕЗЕРВ: 4 1 0 2 -1 -1 -1

08.09.2003 15:01:30: ТЕРМИНАЛ ОСИ : 1 - нет данных

08.09.2003 15:01:34: СКЛАД : 0 - команда ПОЛОЖИТЬ В РЕЗЕРВ успешно принята

08.09.2003 15:01:38: СИСТЕМА : Запрошено сообщение от СКЛАДА

08.09.2003 15:01:38: СКЛАД : 1 - ПОДШИПНИК успешно ПРИНЯТ: 4 1 0 2 -1 -1 -1

0 8 . 0 9 . 2 0 0 3

15:01:38: СИСТЕМА : Запрошен статус СКЛАДА

08.09.2003 15:01:38: СКЛАД : Статус СКЛАДА = 16

Каждая строчка такого вида формируется в кадр. Кадр - набор действий, происходящих в рамках одного события.

Практическая работа №17.
Автоматическая генерация тестов на основе формального описания.

Тесты составляются на основе спецификации требований. При формулировании требований на естественном языке существует проблема их различных толкований. Одним из способов избежать этого является применение формальных языков для описания структуры и поведения системы (UML, SDL, MSC). Кроме того, описание требований на формальном языке является формальным описанием тестовых случаев, на основе которого можно генерировать тестовый код. В практикуме для создания тестов будет использоваться язык диаграмм взаимодействия (Message Sequence Charts, MSC - п.11). В этом случае под тестом мы будем понимать его представление в виде MSC- диаграммы.

В Практикуме для реализации тестирования используется учебная система автоматизации тестирования TAT - Test Automation Training. На вход система принимает формальное описание тестов в виде MSC диаграмм (в текстовом формате MSC PR). На основе этих MSC диаграмм и конфигурационного файла (в формате XML), который описывает интерфейс тестируемой системы, генерируется тест на C#. (Интерфейс тестируемого приложения (Application Under Test - AUT) содержит сигналы, сообщения, транзакции, которые система может посылать тестовому окружению или может принимать от тестового окружения). Для запуска системы с этим тестом необходимо написать Wrapper, который транслирует сигналы от теста к системе и наоборот.

Таким образом, методика тестирования системы с помощью TAT выглядит следующим образом:

- написать Wrapper для тестируемой системы;
- создать файл конфигурации;
- создать формальное описание тестов в виде MSC-диаграмм;
- нарисовать MSC-диаграммы в MS Visio;
- сгенерировать с помощью макроса тестовый файл в формате MPR;
- настроить в ConfigTAT проект теста (указать пути) или набора тестов;
- запустить тест или набор тестов;
- проанализировать получаемые log-файлы.

В данном случае wrapper и файл конфигурации (первые два пункта методики) уже созданы, поэтому вам необходимо будет выполнить только п. 3-6.

В рассматриваемом подходе не только запуск тестов и проверка результатов прогона тестового случая будут осуществляться автоматически, но и сам тестовый код будет генерироваться автоматически на основе MSC-диаграммы (рис. 7.1).

При разработке тестов был использован следующий подход:

Когда реализуется определенное событие, модель посылает сигнал запроса состояния к тестовому окружению.

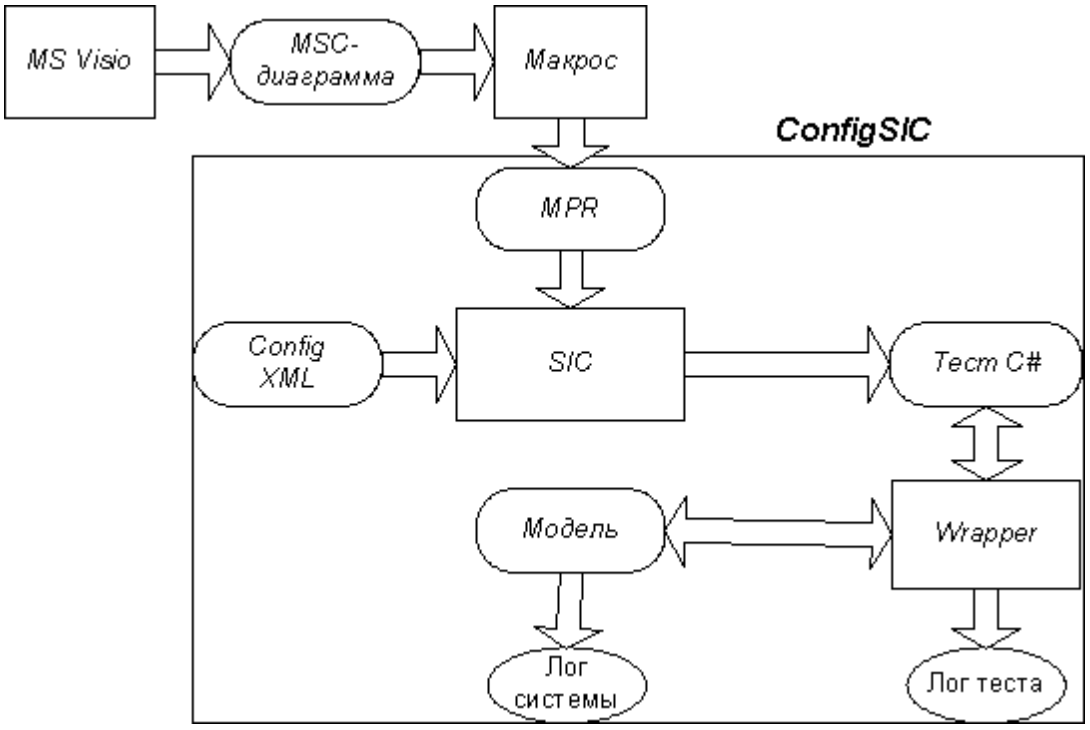


Рис. 7.1. Система и ее окружение (автоматическая генерация)

Состояние окружения задается в тесте (входные данные) в виде параметров сигналов. Тест возвращает состояние окружения (StoreStat, AxlePar, RollerPar, StoreMessage, CommandStatus), посылая модели сигнал с параметрами в соответствии с запросом.

Получаемая информация сохраняется в журнале теста.

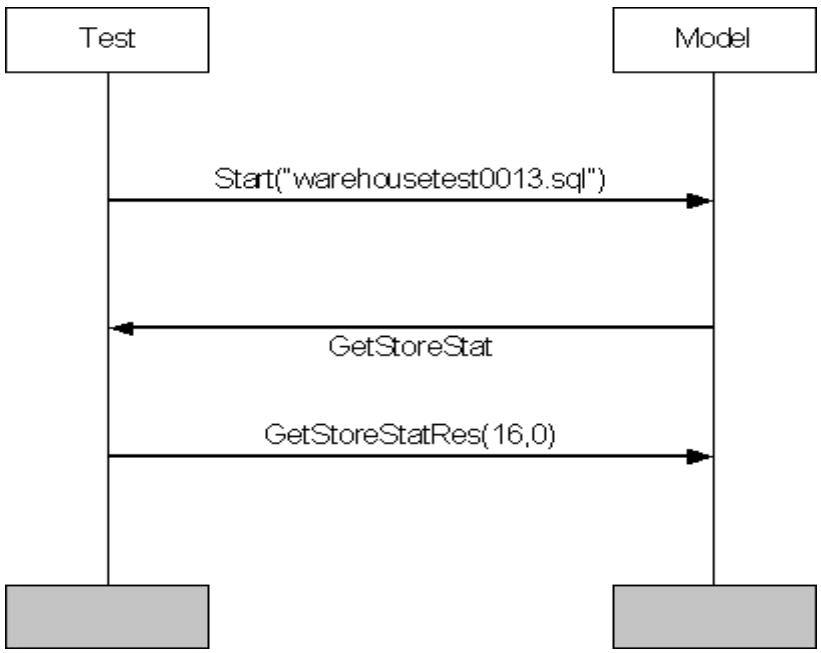


Рис. 7.2. Взаимодействие теста и модели (MSC-диаграмма)

Практическая работа №18.

Автоматизация тестирования с помощью скриптов. Описание тестовых процедур.

Описание тестовых

процедур Как запустить

тест

Запустить run.bat. В файле run.bat запускается tests.bat. Файл tests.bat содержит команды

запуска тестов и установки необходимого состояния базы данных:

osql -H <Host > -S <Server > -d WarehouseTCL -U sa -P sa -i sql\ClearDB.sql

//Вызывается скрипт ClearDB.sql из подкаталога sql.

//Предполагается, что SQL Server выполняется на машине
//<Host> и называется <Server>.

//Эти параметры были настроены автоматически при //установке
практикума.

//База данных называется WarehouseTCL. Если названия отличаются,

//необходимо заменить параметры командной строки утилиты OSQL:

// -h <host> - задает имя хоста, на котором выполняется SQL Server;

// -s <server> - имя сервера;

// -d <db> - имя базы данных;

// -u <username> - имя пользователя; //
-p <password> - пароль;

// -i <script> - имя скрипта SQL.

bin\launcher tests\warehousetest0001.tcl

//Вызывается тест

warehousetest0001.tcl copy log.txt logs\
warehousetest0001.txt

//Файл log.txt (лог тестируемой системы) копируется в файл

//warehousetest0001.txt. Для исключения части тестов из набора

//достаточно закомментировать соответствующие строки (команда
//REM).

Основная литература:

1. Абрамов, Г. В. Проектирование и разработка информационных систем: учебное пособие для СПО / Г. В. Абрамов, И. Е. Медведкова, Л. А. Коробова. — 2-е изд. — Саратов: Профобразование, 2024. — 169 с. — ISBN 978-5-4488-2259-9. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROFобразование: [сайт]. — URL: https://profspo.ru/books/143685

2. Антонов, В.Ф. Методы и средства проектирования информационных систем: учебное пособие / В.Ф. Антонов, А.А. Москвитин; Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северо-Кавказский федеральный университет», Министерство образования и науки Российской Федерации. - Ставрополь: СКФУ, 2016. - 342 с.: ил. - Библиогр. в кн.; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=458663

3. Поляков, М. В. Тестирование программного обеспечения: учебное пособие / М. В. Поляков. — Москва: Ай Пи Ар Медиа, 2023. — 95 с. — ISBN 978-5-4497-2202-7. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROFобразование: [сайт]. — URL: https://profspo.ru/books/130526

Дополнительная литература:

1. Тимофеев, А. В. Проектирование и разработка информационных систем: учебное пособие для СПО / А. В. Тимофеев, З. Ф. Камальдинова, Н. С. Агафонова. — Саратов: Профобразование, 2022. — 91 с. — ISBN 978-5-4488-1416-7. — Текст: электронный // ЭБС PROFобразование: [сайт]. — URL: https://profspo.ru/books/116285

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Северо-Кавказский федеральный университет»

Колледж СКФУ в г. Ставрополе

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к самостоятельной работе

ПМ.03 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНФОРМАЦИОННЫХ СИСТЕМ

Специальность 09.02.11 Разработка и управление программным обеспечением

Форма обучения очная

Ставрополь

1. Пояснительная записка

Методические указания для самостоятельной работы студентов при изучении ПМ.03: МДК.03.01 Проектирование информационных систем, МДК.03.02 Разработка кода информационных систем, МДК.03.03 Сопровождение информационных систем предназначены для обучающихся по специальности 09.02.11 Разработка и управление программным обеспечением.

При организации СР важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

По учебному плану специальности 09.02.11 Разработка и управление программным обеспечением в МДК.03.01 Проектирование информационных систем, МДК.03.02 Разработка кода информационных систем, МДК.03.03 Сопровождение информационных систем на внеаудиторную самостоятельную работу обучающихся в сумме отводится 72 часа.

2. План-график выполнения СРС

№ п/п	Наименование разделов и тем дисциплины, их краткое содержание; вид самостоятельной работы	Форма контроля	Объем часов
МДК.03.01 Проектирование информационных систем			
1.	**Тема 1.13** Сущность функционального подхода к моделированию бизнес процессов. Объектно-ориентированный подход.	Работа с учебной литературой	10
2.	**Тема 1.17** Стадии и этапы создания автоматизированных систем. Виды и наименование проектных документов.	Проверка конспекта	10
МДК.03.02 Разработка кода информационных систем			
1.	**Тема 2.1** Введение в С++. Из истории жизни языка С++. Основы языка С++. Алфавит языка.	Работа с учебной литературой	10
2.	**Тема 2.20** Создание приложения по реализации линейного алгоритма. Структура. Схема и вычисление.	Проверка конспекта	10
МДК.03.03 Сопровождение информационных систем			
1.	**Тема 3.2** Организация и документация процесса внедрения информационных систем	Проверка конспекта	20
2.	**Тема 3.5** Оценки сложности тестирования и методика тестирования объектно-ориентированной программы	Проверка конспекта	12
		Итого:	**72**

3. Методические рекомендации по выполнению самостоятельной работы

3.1. Методические рекомендации по работе с учебной литературой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Основные виды систематизированной записи прочитанного:

Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;
2. Выделите главное, составьте план;
3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;
4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.
5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной

самостоятельной работы.

Информационное обеспечение обучения

Перечень рекомендуемой основной и дополнительной литературы, Интернет-ресурсов, необходимых для освоения профессионального модуля.

Основная литература:

1. Абрамов, Г. В. Проектирование и разработка информационных систем: учебное пособие для СПО / Г. В. Абрамов, И. Е. Медведкова, Л. А. Коробова. — 2-е изд. — Саратов: Профобразование, 2024. — 169 с. — ISBN 978-5-4488-2259-9. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROFобразование : [сайт]. — URL: https://profspo.ru/books/143685

2. Антонов, В.Ф. Методы и средства проектирования информационных систем: учебное пособие / В.Ф. Антонов, А.А. Москвитин; Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северо-Кавказский федеральный университет», Министерство образования и науки Российской Федерации. - Ставрополь: СКФУ, 2016. - 342 с. : ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=458663

3. Поляков, М. В. Тестирование программного обеспечения: учебное пособие / М. В. Поляков. — Москва: Ай Пи Ар Медиа, 2023. — 95 с. — ISBN 978-5-4497-2202-7. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROFобразование : [сайт]. — URL: https://profspo.ru/books/130526

Дополнительная литература:

1. Тимофеев, А. В. Проектирование и разработка информационных систем: учебное пособие для СПО / А. В. Тимофеев, З. Ф. Камальдинова, Н. С. Агафонова. — Саратов: Профобразование, 2022. — 91 с. — ISBN 978-5-4488-1416-7. — Текст: электронный // ЭБС PROFобразование: [сайт]. — URL: https://profspo.ru/books/116285

Интернет-ресурсы:

http://biblioclub.ru/index.php?page=book&id=428825 – Введение в генерацию программного кода

http://www.iprbookshop.ru/61485.html - Кодирование в системах защиты информации

http://biblioclub.ru/index.php?page=book&id=429034 - Объектно-ориентированное программирование и программная инженерия