


МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«БАЙКАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»  
ЧИТИНСКИЙ ИНСТИТУТ  
КОЛЛЕДЖ

УТВЕРЖДАЮ:  
Первый заместитель директора  
  
Н.В. Раевский  
«25» июня 2024 г.

# **ФОНД ОЦЕНОЧНЫХ СРЕДСТВ**

**ПО ПМ 02**  
**ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ**  
МДК.02.02. Инструментальные средства разработки программного обеспечения  
09.02.07 Информационные системы и программирование

Чита 2024

## Структура фонда оценочных средств

### МДК.02.02. Инструментальные средства разработки программного обеспечения по специальности 09.02.07 Информационные системы и программирование

Код компетенции	Умения и знания	Основные показатели оценки результата	Методы оценки
<b>Умения:</b>			
ПК 2.2.	<b>Знания:</b> Модели процесса разработки программного обеспечения. Основные принципы процесса разработки программного обеспечения. Основные подходы к интегрированию программных модулей. Основы верификации программного обеспечения. Современные технологии и инструменты интеграции. Основные протоколы доступа к данным. Методы и способы идентификации сбоев и ошибок при интеграции приложений. Основные методы отладки. Методы и схемы обработки исключительных ситуаций. Основные методы и виды тестирования программных продуктов. Стандарты качества программной документации. Основы организации инспектирования и верификации. Приемы работы с инструментальными средствами тестирования и отладки. Методы организации работы в команде разработчиков.	Задания самостоятельной работы	76% правильных ответов
	<b>Умения:</b> Использовать выбранную систему контроля версий. Использовать методы для получения кода с заданной функциональностью и степенью качества. Организовывать заданную интеграцию модулей в программные средства на базе имеющейся архитектуры и автоматизации бизнес-процессов. Использовать различные транспортные протоколы и стандарты форматирования сообщений. Выполнять тестирование интеграции. Организовывать постобработку данных. Создавать классы-исключения на основе базовых классов. Выполнять ручное и автоматизированное тестирование программного модуля.	Лабораторная работа	Экспертное наблюдение за ходом выполнения лабораторной работы, результат выполнения работы не менее 76%

	Выявлять ошибки в системных компонентах на основе спецификаций. Использовать приемы работы в системах контроля версий.		
ПК 2.3.	<p><b>Знания:</b>          Модели процесса разработки программного обеспечения.          Основные принципы процесса разработки программного обеспечения.          Основные подходы к интегрированию программных модулей.          Основы верификации и аттестации программного обеспечения.          Методы и способы идентификации сбоев и ошибок при интеграции приложений.          Основные методы отладки.          Методы и схемы обработки исключительных ситуаций.          Приемы работы с инструментальными средствами тестирования и отладки.          Стандарты качества программной документации.          Основы организации инспектирования и верификации.          Встроенные и основные специализированные инструменты анализа качества программных продуктов.          Методы организации работы в команде разработчиков.</p>	Экзамен	Оценка результатов экзамена 76% правильных ответов
	<p><b>Умения:</b>          Использовать выбранную систему контроля версий.          Использовать методы для получения кода с заданной функциональностью и степенью качества.          Анализировать проектную и техническую документацию.          Использовать инструментальные средства отладки программных продуктов.          Определять источники и приемники данных.          Выполнять тестирование интеграции.          Организовывать постобработку данных.          Использовать приемы работы в системах контроля версий.          Выполнять отладку, используя методы и инструменты условной компиляции.          Выявлять ошибки в системных компонентах на основе спецификаций</p>	Лабораторные работы	Экспертное наблюдение за ходом выполнения лабораторных работ, результат выполнения работы не менее 76%
ПК 2.5.	<p><b>Знания:</b>          Модели процесса разработки программного обеспечения.          Основные принципы процесса разработки</p>	Тестирование	76% правильных ответов

	<p>программного обеспечения.</p> <p>Основные подходы к интегрированию программных модулей.</p> <p>Основы верификации и аттестации программного обеспечения.</p> <p>Стандарты качества программной документации.</p> <p>Основы организации инспектирования и верификации.</p> <p>Встроенные и основные специализированные инструменты анализа качества программных продуктов.</p> <p>Методы организации работы в команде разработчиков.</p>		
	<p><b>Умения:</b></p> <p>Использовать выбранную систему контроля версий.</p> <p>Использовать методы для получения кода с заданной функциональностью и степенью качества.</p> <p>Анализировать проектную и техническую документацию.</p> <p>Организовывать постобработку данных.</p> <p>Приемы работы в системах контроля версий.</p> <p>Выявлять ошибки в системных компонентах на основе спецификаций.</p>	Лабораторные работы	<p>Экспертное наблюдение за ходом выполнения лабораторной работы, результат выполнения работы не менее 76%</p>

**Структура фонда оценочных средств**  
**по МДК.02.02. Инструментальные средства разработки программного обеспечения**

№п/п	Проверяемые умения, знания, ОК, ПК	Тема из рабочей программы	Наименование оценочного средства	Критерии оценивания
1.	ПК 2.2, ПК 2.3, ПК 2.5	Тема 2.1. Современные технологии и инструменты интеграции.	<p>1. Лабораторная работа «Разработка модульной структуры проекта (диаграммы модулей)»</p> <p>2. Лабораторная работа «Настройка работы системы контроля версий (типов импортируемых файлов, путей, фильтров и др. параметров импорта в репозиторий)»</p> <p>3. Лабораторная работа «Разработка и интеграция модулей проекта (командная работа)»</p> <p>4. Лабораторная работа «Отладка отдельных модулей программного проекта»</p>	<p>0-4 - «2»</p> <p>5-6 - «3»</p> <p>7-8 - «4»</p> <p>9-10 - «5»</p>
2.	ПК 2.2, ПК 2.3, ПК 2.5	Тема 2.2. Инструментарий тестирования и анализа качества программных средств	<p>1. Лабораторная работа «Отладка проекта»</p> <p>2. Лабораторная работа «Инспекция кода модулей проекта»</p> <p>3. Лабораторная работа «Тестирование интерфейса пользователя средствами инструментальной среды разработки»</p> <p>4. Лабораторная работа «Разработка тестовых модулей проекта для тестирования отдельных модулей»</p> <p>5. Лабораторная работа «Выполнение функционального тестирования»</p>	<p>0-4 - «2»</p> <p>5-6 - «3»</p> <p>7-8 - «4»</p> <p>9-10 - «5»</p>

			6. Лабораторная работа «Тестирование интеграции» 7. Лабораторная работа «Документирование результатов тестирования»	
4			Устный зачет	0-7 - «2» 8-10 - «3» 11-13 - «4» 14-15 - «5» «5» – верный, подробный ответ; «4» – одна ошибка или не более двух недочетов; «3» – две ошибки или не более трех недочетов; «2» – больше двух ошибок или трех недочетов
				0-10 - «2» 11-14 - «3» 15-17 - «4» 18-20 - «5»
6		Итоговая аттестация за 2 семестр	Устный зачет	«5» – верный, подробный ответ; «4» – одна ошибка или не более двух недочетов; «3» – две ошибки или не более трех недочетов; «2» – больше двух ошибок или трех недочетов.
7		Итоговая аттестация по ПМ	Экзамен	«5» – верный, подробный ответ; «4» – одна ошибка или не более двух недочетов; «3» – две ошибки или не более трех недочетов; «2» – больше двух ошибок или трех недочетов

## **Лабораторная работа «Разработка модульной структуры проекта (диаграммы модулей)»**

### **Разработка эскизного проекта**

Эскизный проект возникает как результат анализа требований, предъявленных к программному продукту. В нем в общем виде формулируются указания по созданию программного продукта. Здесь ставится задача для каждого разработчика, описываются алгоритм решения задачи, способы взаимодействия создаваемого продукта с другими программами и устройствами ввода-вывода, выбираются структуры данных, определяются способы хранения данных на диске или в базе данных.

Эскизный проект не может быть слишком большим. Он должен быть обозримым, схематичным, четко показывающим основные этапы создания программного продукта. Обычно эскизный проект содержит не больше 5—6 страниц текста. К нему прилагаются диаграммы, рисунки и чертежи, а также календарный план выполнения проекта.

После того как эскизный проект создан, он раздается всем участникам разработки для изучения и обсуждения. Каждый разработчик обдумывает свой участок проекта, вносит свои предложения и дополнения, конкретизирует план выполнения проекта.

### **Разработка технического проекта**

После изучения эскизного проекта всеми заинтересованными лицами наступает время создания технического проекта. В его обсуждении принимает участие вся команда разработчиков под руководством менеджера проекта. Каждый разработчик вносит свои предложения по реализации и улучшению проекта, уточняет и детализирует относящиеся к нему положения проекта, согласует интерфейсы с другими разработчиками.

Технический проект будет рабочим документом на все время реализации проекта, поэтому он должен быть понятен и приемлем для всех программистов. В нем не должно быть недомолвок, двусмысленностей, не должно оставаться пробелов и недоговоренностей.

При разработке технического проекта окончательно определяется конфигурация технических средств, и вся дальнейшая работа ведется с учетом этой конфигурации. Уточняется операционная среда, в которой будет функционировать программный продукт, и системное программное обеспечение. Например, Web-приложение работает в браузере. Браузеры по-разному интерпретируют языки HTML и JavaScript, поэтому надо сразу решить, будет ли программный продукт рассчитан на определенный браузер или он должен работать в любом. В первом случае разработчики могут включить в продукт дополнительные возможности языков HTML и JavaScript, интерпретируемые данным браузером, во втором — должны использовать только стандартные конструкции, что может значительно затруднить разработку.

В техническом проекте уточняются типы и структуры исходных и промежуточных данных, полностью детализируется алгоритм решения задачи. Задача разбивается на модули, которые распределяются среди программистов.

При объектно-ориентированном проектировании в техническом проекте определяются все объекты, необходимые для осуществления проекта и выявляются связи между ними. Полностью выписывается строение каждого объекта, его поля и методы. Объекты записываются в виде интерфейсов или абстрактных классов, дальнейшая разработка которых поручается конкретным программистам.

После проработки технического проекта каждым участником разработки собираются и обобщаются их уточнения и замечания. Окончательная версия проекта обсуждается командой разработчиков. Менеджер проекта выносит технический проект на утверждение руководством фирмы-разработчика и заказчиком программного продукта. После этого технический проект становится рабочим проектом для группы разработчиков.

## **Рабочий проект**

После утверждения технического проекта он становится основным рабочим документом для команды разработчиков программного продукта. Рабочий проект — это большой, подробный документ, наиболее полно описывающий будущий программный продукт и план его создания. В нем содержатся детальные указания каждому разработчику и команде в целом, определена структура базы данных и других хранилищ данных, которой будут руководствоваться все разработчики. Короче говоря, в рабочем проекте должны содержаться все сведения, нужные каждому разработчику и команде в целом. В частности, в нем должны быть записаны этапы и сроки разработки, чтобы каждый программист твердо знал их.

При объектно-ориентированном проектировании в рабочем проекте должны быть полностью описаны все классы и связи между ними. Это описание можно сделать в виде абстрактных классов или интерфейсов, на языке разработки или на языке описания. Важно, чтобы все участники проекта правильно понимали эту запись и одинаково интерпретировали ее.

Каждому участнику проекта выдается экземпляр рабочего проекта. При всяком изменении рабочего проекта участники получают его новую версию. В настоящее время с развитием Web-технологии, как правило, создается собственный сайт для каждого проекта. Все рабочие документы публикуются на этом сайте, а при каждом их изменении участники проекта получают уведомление по электронной почте.

### **Упражнения**

1. Разработайте проект автоматизации библиотечного каталога.
2. Проведите анализ работы деканата и разработайте проект его автоматизации.
3. Проанализируйте информационные потоки вашего факультета и спроектируйте компьютерную систему их обработки.

## **Лабораторная работа «Настройка работы системы контроля версий (типов импортируемых файлов, путей, фильтров и др. параметров импорта в репозиторий)»**

Система управления/контроля версиями (от англ. Version Control System или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости, возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение и многое другое.

Такие системы наиболее широко применяются при разработке программного обеспечения, для хранения исходных кодов разрабатываемой программы. Однако, они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов, в частности, они всё чаще применяются в САПР, обычно, в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).

Распространённые системы управления версиями

- Subversion
- Darcs
- Microsoft Visual SourceSafe
- Bazaar
- Rational ClearCase
- Perforce



- BitKeeper
- Mercurial
- Git
- GNU Arch
- CVS — устаревшая. Потомок: Subversion
- RCS — устаревшая. Потомок: CVS

#### Основные понятия

Репозиторий (repository) – центральное хранилище, которое содержит версии файлов. Очень часто репозиторий организуется средствами какой-нибудь СУБД.

Версия файла (revision) – состояние файла в определенный момент времени. Репозиторий предоставляет возможность хранить неограниченное число версий одного и того же файла.

Актуальная версия файла – обычно это самая последняя версия файла, размещенного в репозитории.

Рабочая версия файла (working copy) – версия файла, с которой в текущий момент ведется работа, и которая не загружена в репозиторий.

Загрузка (Upload) – размещение файла в репозитории. В процессе загрузки в репозиторий помещается рабочая версия файла.

Выгрузка (Checkout) – получение файла из репозитория. В процессе выгрузки осуществляется получение из репозитория необходимой версии файла.

Синхронизация (update, sync) – приведение в соответствие рабочих версий файлов с актуальными версиями в репозитории. В процессе синхронизации в репозиторий загружаются те файлы, рабочие копии которых являются более "свежими" (т.е. имеют более поздние версии), по сравнению с файлами в репозитории, и выгружаются те файлы, рабочие копии которых устарели по сравнению с копиями в репозитории.

#### Borland StarTeam

Borland StarTeam – очень мощный и функциональный кросс-платформенный продукт, разрабатываемый в прошлом фирмой StarBase, которую Borland приобрела в конце 2002 г.

Заметное преимущество данного решения состоит в том, что версия 2005 выступает центральным элементом стратегии управления жизненным циклом приложений (Application Lifecycle Management, ALM) компании Borland и обладает расширенными возможностями интеграции со всеми ее ключевыми пакетами, используемыми при разработке программного обеспечения.

#### MS SourceSafe

Microsoft Visual SourceSafe (Visual SourceSafe, VSS) — программный продукт компании Майкрософт, файл-серверная система управления версиями, предназначенная для небольших команд разработчиков. VSS позволяет хранить в общем хранилище файлы, разделяемые несколькими пользователями, для каждого файла хранится история версий. VSS входит в состав пакета Microsoft Visual Studio и интегрирован с продуктами этого пакета. Доступен только для платформы Windows. Версию для Unix поддерживает компания MainSoft. В ноябре 2005 года вышла обновлённая версия продукта — Visual SourceSafe 2005, обещающая повышенную стабильность и производительность, улучшенный механизм слияния для XML-файлов и файлов в Юникоде, а также работу через HTTP. Visual SourceSafe нацелен на индивидуальных разработчиков либо небольшие команды разработчиков. Там где VSS недостаточно, ему на замену предлагается новый продукт Майкрософт — Team Foundation Server, входящий в состав Visual Studio Team System.

#### Rational Clear Case

ClearCase поддерживает следующие возможности, разительно отличающие его в лучшую сторону от других средств контроля:

- Общий контроль версий не только файлов, но и директорий/поддиректорий;
- Бесконечное число ответвлений от определенной версии;
- Автоматическая компрессия файлов и их кеширование (СС позволяет хранить большое количество данных, при всем при этом база данных остается компактной и быстрой);
- Позволяет легко конвертировать базы данных других средств контроля, например: PVCS, SourceSafe, RCS, CVS и SCCS;
- Поддерживает параллельную разработку и мультикомандные подразделения, расположенные в географически удаленных друг от друга местах;
- Мультиплатформенность (способен объединить единой средой участников, работающих на разных операционных системах);
- Имеет интеграцию со средствами разработки;
- Имеет Web-интерфейс для удаленного контроля.

#### CVS

CVS (Concurrent Versions System, "Система Конкурирующих Версий" ). Хранит историю изменений определённого набора файлов, как правило исходного кода программного обеспечения, и облегчает совместную работу группы людей (часто — программистов) над одним проектом. CVS популярна в мире открытого ПО. Система распространяется на условиях лицензии GNU GPL.

#### Subversion

Subversion — централизованная система (в отличие от распределённых систем, таких, как Git или Mercurial), то есть данные хранятся в едином хранилище. Хранилище может располагаться на локальном диске или на сетевом сервере. Работа в Subversion мало отличается от работы в других централизованных системах управления версиями. Для совместной работы над файлами в Subversion преимущественно используется модель Копирование-Изменение-Слияние. Кроме того, для файлов, не допускающих слияние (различные бинарные форматы файлов), можно использовать модель Блокирование-Изменение-Разблокирование.

1. Настроить подключение к репозиторию
2. Скачать проект
3. Добавить свой класс к проекту
4. Внести изменения к класс
5. Обновить класс в репозитории
6. Удалить все локальные файлы и скачать проект из репозитория
7. Добавить "лишний" файл в репозиторий и затем удалить его из репозитория.
8. Изучить журнал изменений файлов, посмотреть какие изменения внесены другими разработчиками.

Примечание: опробовать Git, Subversion, Mercurial (локально)

## Лабораторная работа «Разработка и интеграция модулей проекта (командная работа)»

**Цель работы.** Освоить процесс проектирования модулей программного обеспечения.

#### Задание.

1. Описать этапы проектирования модулей программы.
2. Составить в виде блок-схемы алгоритм решения задачи.
3. Составить отчет по практической работе.

**Отчет по практической работе должен включать:**

1. Алгоритм решения задачи.

2. Набор тестов для отладки программы.

**Задача.** Составить алгоритм решения задачи, приведенной ниже, с использованием структурных единиц: процедур и/или функций.

**Варианты индивидуальных заданий.**

1. Даны два двумерных массива вещественных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать номера столбцов, содержащих только положительные элементы. Если таковых столбцов в массиве нет, то вывести соответствующее сообщение. Проверку столбца на положительность элементов оформить в виде процедуры с передачей в нее всех элементов текущего столбца.

2. Даны два двумерных массива натуральных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать номера столбцов, содержащих только кратные 5 или 7 элементы. Если таких столбцов в массиве нет, то вывести соответствующее сообщение. Проверку столбца на наличие указанных элементов оформить в виде процедуры с передачей в нее всех элементов текущего столбца.

3. Даны пять одномерных массива вещественных элементов. Размер каждого массива не превосходит 100 элементов. Для каждого из массивов определить, составляют ли его элементы знакочередующуюся последовательность. Если да, то указать порядковый номер такого массива, в противном случае вывести отрицательный ответ. Проверку массива на выполнение условия оформить в виде процедуры с передачей в нее всех элементов рассматриваемого массива.

4. Даны два двумерных массива символьных (буквы русского алфавита) элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать номера строк, содержащих элементы только строчных букв, если таких строк нет ни для какого массива, то вывести соответствующее сообщение. Проверку строки на наличие указанных элементов оформить в виде процедуры с передачей в нее всех элементов текущей строки.

5. Даны два двумерных массива вещественных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать количество столбцов, содержащих только не положительные элементы. Если таких столбцов нет ни для одного из массивов, то вывести соответствующее сообщение. Проверку столбца на наличие указанных элементов оформить в виде процедуры с передачей в нее всех элементов текущего столбца.

6. Даны пять одномерных массива вещественных элементов. Размер каждого массива не превосходит 100 элементов. Для каждого из массивов определить, составляют ли его элементы одного знака. Если да, то указать порядковый номер такого массива, в противном случае вывести отрицательный ответ. Проверку массива на выполнение условия оформить в виде процедуры с передачей в нее всех элементов рассматриваемого массива.

7. Даны два двумерных массива целочисленных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать количество строк, содержащих элементы, четность которых чередуется, а вторым в четных строках является нечетный элемент. Если таких строк нет ни для одного из массивов, то вывести соответствующее сообщение. Проверку строки на наличие указанных элементов оформить в виде процедуры с передачей в нее всех элементов текущего столбца.

8. Даны пять одномерных массива символьных (только латинские буквы) элементов. Размер каждого массива не превосходит 100 элементов. Для каждого из массивов определить, чередуются ли в нем буквы строчные и прописные. Если да, то указать порядковый номер такого массива, в противном случае вывести отрицательный ответ. Проверку массива на выполнение условия оформить в виде процедуры с передачей в нее всех элементов рассматриваемого массива.

9. Даны два двумерных массива целочисленных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать количество строк, для которых сумма элементов, стоящих на нечетных местах в строке, является положительным числом. Если таких строк нет ни для одного из массивов, то вывести соответствующее сообщение. Проверку строки на выполнение условия и расчет оформить в виде процедуры с передачей в нее всех элементов текущей строки.

10. Даны два двумерных массива вещественных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать номера столбцов, произведение отрицательных элементов которых является положительным числом. Если таких столбцов нет ни для одного из массивов, то вывести соответствующее сообщение. Проверку столбца на выполнение условия и расчет оформить в виде процедуры с передачей в нее всех элементов текущего столбца.

11. Даны пять одномерных массива символьных (только латинские буквы) элементов. Размер каждого массива не превосходит 100 элементов. Для каждого из массивов определить, расположены ли в нем строчные буквы в алфавитном порядке. Если да, то указать порядковый номер такого массива, в противном случае вывести отрицательный ответ. Проверку массива на выполнение условия оформить в виде процедуры с передачей в нее всех элементов рассматриваемого массива.

12. Даны два двумерных массива целочисленных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов проверить выполнение условия: все четные строки массива таковы, что суммы их элементов образуют возрастающую последовательность. Вывести соответствующее сообщение. Вычисление суммы элементов массива и проверку последовательности чисел на выполнение условия оформить в виде процедуры с передачей в нее всех необходимых элементов.

13. Даны два двумерных массива вещественных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Преобразовать все нечетные строки каждого массива так, чтобы элементы составляли возрастающую по абсолютной величине последовательность. Вывести преобразованные массивы. Упорядочивание элементов оформить в виде процедуры с передачей в нее всех необходимых элементов.

14. Даны два двумерных массива целочисленных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого столбца массивов вычислить суммы и количества элементов, значения которых находятся в заданном диапазоне. Если чисел, удовлетворяющих этому условию нет, то вывести соответствующее сообщение. Вычисление для элементов столбца массива оформить в виде процедуры с передачей в нее всех необходимых элементов.

15. Даны пять одномерных массива символьных (только латинские буквы) элементов. Размер каждого массива не превосходит 100 элементов. Преобразовать все массивы так, чтобы все строчные буквы были расположены по алфавиту. При этом переставлять только строчные буквы, оставив прописные буквы на своих местах. Преобразование каждого массива оформить в виде процедуры с передачей в нее всех необходимых элементов. Если перестановка элементов не потребовалась, то есть исходные массивы удовлетворяют требуемому условию, то вывести соответствующее сообщение.

#### **Задание.**

1. Разработать модули программы, спроектированные во время практического занятия
2. Отладить программу с использованием тестов, составленных во время практического занятия
3. Составить отчет по практической работе.

## Лабораторная работа «Отладка отдельных модулей программного проекта»

**Цель работы.** Изучить основные подходы к проектированию тестов.

Теоретическая часть.

Рассмотрим два основных подхода к проектированию тестов.

Первый подход ориентируется только на стратегию тестирования, называемую стратегией "черного ящика", тестированием с управлением по данным или тестированием с управлением по входу-выходу. При использовании этой стратегии программа рассматривается как черный ящик. Тестовые данные используются только в соответствии со спецификацией программы (т. е. без учета знаний о ее внутренней структуре). Недостижимый идеал сторонника первого подхода — проверить все возможные комбинации и значения на входе. Обычно их слишком много даже для простейших алгоритмов. Так, для программы расчета среднего арифметического четырех чисел надо готовить  $10^7$  тестовых данных.

При первом подходе обнаружение всех ошибок в программе является критерием исчерпывающего входного тестирования. Последнее может быть достигнуто, если в качестве тестовых наборов использовать все возможные наборы входных данных. Следовательно, приходим к выводу, что для исчерпывающего тестирования программы требуется бесконечное число тестов, а значит, построение исчерпывающего входного теста невозможно. Это подтверждается двумя аргументами: во-первых, нельзя создать тест, гарантирующий отсутствие ошибок; во-вторых, разработка таких тестов противоречит экономическим требованиям. Поскольку исчерпывающее тестирование исключается, нашей целью должна стать максимизация результативности капиталовложений в тестирование (максимизация числа ошибок, обнаруживаемых одним тестом). Для этого необходимо рассматривать внутреннюю структуру программы и делать некоторые разумные, но, конечно, не обладающие полной гарантией достоверности предположения.

Второй подход использует стратегию "белого ящика", или стратегию тестирования, управляемую логикой программы, которая позволяет исследовать внутреннюю структуру программы. В этом случае тестировщик получает тестовые данные путем анализа только логики программы; стремится, чтобы каждая команда была выполнена хотя бы один раз. При достаточной квалификации добивается, чтобы каждая команда условного перехода выполнялась бы в каждом направлении хотя бы один раз. Цикл должен выполняться один раз, ни разу, максимальное число раз. Цель тестирования всех путей извне также недостижима. В программе из двух последовательных циклов внутри каждого из них включено ветвление на десять путей, имеется  $10^{18}$  путей расчета. Причем выполнение всех путей расчета не гарантирует выполнения всех спецификаций.

Сравним способ построения тестов при данной стратегии с исчерпывающим входным тестированием стратегии "черного ящика". Неверно предположение, что достаточно построить такой набор тестов, в котором каждый оператор выполняется хотя бы один раз. Исчерпывающему входному тестированию может быть поставлено в соответствие исчерпывающее тестирование маршрутов. Подразумевается, что программа проверена полностью, если с помощью тестов удастся осуществить выполнение этой программы по всем возможным маршрутам ее потока (графа) передач управления.

Последнее утверждение имеет два слабых пункта: во-первых, число не повторяющихся друг друга маршрутов — астрономическое; во-вторых, даже если каждый маршрут может быть проверен, сама программа может содержать ошибки (например, некоторые маршруты пропущены).

Свойство пути выполняться правильно для одних данных и неправильно для других — называемое чувствительностью к данным, наиболее часто проявляется за счет численных погрешностей и погрешностей усечения методов. Тестирование каждого из всех маршрутов одним тестом не гарантирует выявление чувствительности к данным.

В результате всех изложенных выше замечаний отметим, что ни исчерпывающее входное тестирование, ни исчерпывающее тестирование маршрутов не могут стать полезными стратегиями, потому что оба они нереализуемы. Поэтому реальным путем, который позволит создать хорошую, но, конечно, не абсолютную стратегию, является сочетание тестирования программы несколькими методами.

Рассмотрим пример тестирования оператора:

*if A and B then...*

при использовании разных критериев полноты тестирования.

При критерии покрытия условий требовались бы два теста:  $A = \text{true}$ ,  $B = \text{false}$  и  $A = \text{false}$ ,  $B = \text{true}$ . Но в этом случае не выполняется then-предложение оператора if.

Существует еще один критерий, названный покрытием решений/условий. Он требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия в решении выполнялись, по крайней мере, один раз; все результаты каждого решения выполнялись тоже один раз и каждой точке входа передавалось управление, по крайней мере, один раз.

Недостатком критерия покрытия решений/условий является невозможность его применения для выполнения всех результатов всех условий. Часто подобное выполнение имеет место вследствие того, что определенные условия скрыты другими условиями. Например, если условие AND есть ложь, то никакое из последующих условий в выражении не будет выполнено. Аналогично, если условие OR есть истина, то никакое из последующих условий не будет выполнено. Следовательно, критерии покрытия условий и покрытия решений/условий недостаточно чувствительны к ошибкам в логических выражениях.

Критерием, который решает эти и некоторые другие проблемы, является комбинаторное покрытие условий. Он требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении и все точки входа выполнялись, по крайней мере, один раз.

В случае циклов число тестов для удовлетворения критерию комбинаторного покрытия условий обычно больше, чем число путей.

Легко видеть, что набор тестов, удовлетворяющий критерию комбинаторного покрытия условий, удовлетворяет также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

Таким образом, для программ, содержащих только одно условие на каждое решение, минимальным является критерий, набор тестов которого вызывает выполнение всех результатов каждого решения, по крайней мере, один раз; передает управление каждой точке входа (например, оператор CASE).

Для программ, содержащих решения, каждое из которых имеет более одного условия, минимальный критерий состоит из набора тестов, вызывающих все возможные комбинации результатов условий в каждом решении и передающих управление каждой точке входа программы, по крайней мере, один раз.

Деление алгоритма на типовые стандартные структуры позволяет минимизировать усилия программиста, затрачиваемые им на тестирование. Запрет на вложенные структуры как раз и объясняется излишними затратами на тестирование. Использование цепочки простых альтернатив с одним действием или структуры ВЫБОР вместо вложенных простых АЛЬТЕРНАТИВ значительно сокращает число тестов!

**Задание.**

1. Оформить внешнюю спецификацию.
2. Составить в виде блок-схемы алгоритм решения задачи.
3. Создать программу решения задачи на любом алгоритмическом языке программирования.
4. Составить набор тестов и провести тестирование созданной программы с помощью методов «белого ящика» (покрытия операторов, покрытия решений, покрытия условий, комбинаторного покрытия условий).

5. Оформить отчет по лабораторной работе.

**Отчет по лабораторной работе должен включать:**

1. Внешнюю спецификацию.
2. Алгоритм решения задачи.
3. Текст программы на языке программирования.
4. Набор тестов для отладки программы, соответствующий конкретным методам «белого ящика».

**Задача.** «Нахождение характерных точек функции». Составить алгоритм и написать программу последовательного вычисления значений заданной функции  $Y(X)$  до тех пор, пока не будет пройдена некоторая характерная точка графика функции. Значения аргумента  $X$  составляют возрастающую последовательность с шагом  $h$ . Начальное значение  $X$  0 и шаг изменения аргумента  $h$  задаются пользователем.

#### Варианты индивидуальных заданий.

##### Вид функции и характерные точки:

0	Локальный максимум функции $\frac{\sqrt{2x^2+1}}{3+x^2}$	5	Точка (точки), в которой функция $\frac{x^2+5}{x^2+7}$ равна 15.
1	Пересечение графиков функций $10\cos(x)$ и $x^3/3+5$ .	6	Все нули функции $\frac{x^2-15}{2x^2+3x-7}$ .
2	Локальный максимум функции $-(x-2)^2+\cos(x)$	7	Пересечение графиков функций $3x^2-15$ и $10\cos(x)+7$ .
3	Точка (точки), в которой функция $3x^2-\frac{12x-5}{x^2+1}$ равна 5.	8	Локальный минимум функции $-100x+e^x$
4	Все нули функции $\frac{x+5}{x^2-7}$ .	9	Локальный максимум функции $-\frac{x}{x^2+3}$
5	Локальный максимум функции $-e^{-x}-100x$	0	Пересечение графиков функций $4(x-5)^2-15$ и $\frac{100}{x^2+2}-7$ .

### Лабораторная работа «Отладка проекта»

**Цель работы.** Получение практических навыков тестирования и отладки программы.

#### Теоретические основы.

Тестирование – процесс выполнения программы на наборе тестов с целью выявления ошибок.

Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки

необходимо определить ее причину, т.е. определить оператор или фрагмент, содержащие ошибку. Причины ошибок могут быть как очевидны, так и очень глубоко скрыты. В целом сложность отладки обусловлена следующими причинами:

- требует от программиста глубоких знаний специфики управления используемыми техническими средствами, операционной системы, среды и языка программирования, реализуемых процессов, природы и специфики различных ошибок, методик отладки и соответствующих программных средств;
- психологически дискомфортна, так как необходимо искать собственные ошибки и, как правило, в условиях ограниченного времени;
- возможно взаимовлияние ошибок в разных частях программы, например, за счет затирания области памяти одного модуля другим из-за ошибок адресации;
- отсутствуют четко сформулированные методики отладки.

Отладка программы в любом случае предполагает обдумывание и логическое осмысление всей имеющейся информации об ошибке. Большинство ошибок можно обнаружить по косвенным признакам посредством тщательного анализа текстов программ и результатов тестирования без получения дополнительной информации. При этом используют различные методы:

- ручного тестирования;
- индукции;
- дедукции;
- обратного прослеживания.

#### ***Метод ручного тестирования***

Это - самый простой и естественный способ данной группы. При обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя тестовый набор, при работе с которыми была обнаружена ошибка. Метод очень эффективен, но не применим для больших программ, программ со сложными вычислениями и в тех случаях, когда ошибка связана с неверным представлением программиста о выполнении некоторых операций. Данный метод часто используют как составную часть других методов отладки.

#### ***Метод индукции***

Метод основан на тщательном анализе симптомов ошибки, которые могут проявляться как неверные результаты вычислений или как сообщение об ошибке. Если компьютер просто "зависает", то фрагмент проявления ошибки вычисляют, исходя из последних полученных результатов и действий пользователя. Полученную таким образом информацию организуют и тщательно изучают, просматривая соответствующий фрагмент программы. В результате этих действий выдвигают гипотезы об ошибках, каждую из которых проверяют. Если гипотеза верна, то детализируют информацию об ошибке, иначе - выдвигают другую гипотезу. Последовательность выполнения отладки методом индукции показана на рисунке в виде схемы алгоритма.

Самый ответственный этап - выявление симптомов ошибки. Организуя данные об ошибке, целесообразно записать все, что известно о её проявлениях, причем фиксируют, как ситуации, в которых фрагмент с ошибкой выполняется нормально, так и ситуации, в которых ошибка проявляется. Если в результате изучения данных никаких гипотез не появляется, то необходима дополнительная информация об ошибке. Дополнительную информацию можно получить, например, в результате выполнения схожих тестов. В процессе доказательства пытаются выяснить, все ли проявления ошибки объясняет данная гипотеза, если не все, то либо гипотеза не верна, либо ошибок несколько.

#### ***Метод дедукции***



По методу дедукции вначале формируют множество причин, которые могли бы вызвать данное проявление ошибки. Затем анализируя причины, исключают те, которые противоречат имеющимся данным. Если все причины исключены, то следует выполнить дополнительное тестирование исследуемого фрагмента. В противном случае наиболее вероятную гипотезу пытаются доказать. Если гипотеза объясняет полученные признаки ошибки, то ошибка найдена, иначе - проверяют следующую причину.

#### ***Метод обратного прослеживания***

Для небольших программ эффективно применение метода обратного прослеживания. Начинают с точки вывода неправильного результата. Для этой точки строится гипотеза о значениях основных переменных, которые могли бы привести к получению имеющегося результата. Далее, исходя из этой гипотезы, делают предложения о значениях переменных в предыдущей точке. Процесс продолжают, пока не обнаружат причину ошибки.

#### **Задание.**

1. Составить в виде блок-схемы алгоритм решения задачи.
2. Создать программу решения задачи на любом алгоритмическом языке программирования.
3. Отладить программу.
4. Составить отчет по лабораторной работе.

#### ***Отчет по лабораторной работе должен включать:***

1. Алгоритм решения задачи.
2. Текст программы на языке программирования.
3. Набор тестов для отладки программы.

**Задача:** составить список учебной группы, включающей 25 человек. Для каждого учащегося указать дату рождения, год поступления в колледж, курс, группу, оценки каждого года обучения.

Назначение задачи: получить значение определённого критерия и упорядочить список студентов по нему.

Достигаемая цель: упорядочить список студентов по среднему баллу и получить его.

## **Лабораторная работа «Тестирование интерфейса пользователя средствами инструментальной среды раз-работки»**

**Цель работы.** Получение практических навыков автоматической генерации тестов на основе формального описания.

Теоретическая часть.

Практически все программные системы предусматривают интерфейс с оператором. Практически всегда этот интерфейс – графический (GUI – Graphical User's Interface). Соответственно, актуальна и задача тестирования создаваемого графического интерфейса.

Вообще говоря, задача тестирования создаваемых программ возникла практически одновременно с самими программами. Известно, что эта задача очень трудоёмка как в смысле усилий по созданию достаточного количества тестов (отвечающих заданному критерию тестового покрытия), так и в смысле времени прогона всех этих тестов. Поэтому решение этой задачи стараются автоматизировать (в обоих смыслах).

Для решения задачи тестирования программ с программным интерфейсом (API – Application Program Interface: вызовы методов или процедур, пересылки сообщений) известны подходы – методы и инструменты – хорошо зарекомендовавшие себя в индустрии создания программного обеспечения. Основа этих подходов следующая: создается формальная спецификация программы, и по этой спецификации генерируются как сами тесты, так и тестовые

оракулы – программы, проверяющие правильность поведения тестируемой программы. Спецификации, как набор требований к создаваемой программе, существовали всегда. Ключевым словом здесь является формальная спецификация. Формальная спецификация – это спецификация в форме, допускающей её формальные же преобразования и обработку компьютером. Это позволяет анализировать набор требований с точки зрения их полноты, непротиворечивости и т.п. Для задачи автоматизации тестирования эта формальная запись должна также обеспечивать возможность описания формальной связи между понятиями, используемыми в спецификации, и сущностями языка реализации программы.

Правильность функционирования системы определяется соответствием реального поведения системы эталонному поведению. Для того чтобы качественно определять это соответствие, нужно уметь формализовать эталонное поведение системы. Распространённым способом описания поведения системы является описание с помощью диаграмм UML (Unified Modeling Language). Стандарт UML предлагает использование трех видов диаграмм для описания графического интерфейса системы:

- Диаграммы сценариев использования (Use Case).
- Диаграммы конечных автоматов (State Chart).
- Диаграммы действий (Activity).

С помощью UML/Use Case diagram можно описывать на высоком уровне наборы сценариев использования, поддерживаемых системой. Данный подход имеет ряд преимуществ и недостатков по отношению к другим подходам, но существенным с точки зрения автоматизации генерации тестов является недостаточная формальная строгость описания.

Широко распространено использование UML/State Chart diagram для спецификации поведения системы, и такой подход очень удобен с точки зрения генерации тестов. Но составление спецификации поведения современных систем с помощью конечных автоматов есть очень трудоёмкое занятие, так как число состояний системы очень велико. С ростом функциональности системы спецификация становится всё менее и менее наглядной.

Перечисленные недостатки этих подходов к описанию поведения системы преодолеваются с помощью диаграмм действий (Activity). С одной стороны нотация UML/Activity diagram является более строгой, чем у сценариев использования, а с другой стороны предоставляет более широкие возможности по сравнению с диаграммами конечных автоматов.

Для создания прототипа работающей версии данного подхода используется инструмент Rational Rose. В первую очередь для спецификации графического интерфейса пользователя при помощи диаграмм действий UML.

Для прогона сгенерированных по диаграмме состояний тестов используется инструмент Rational Robot. Из возможностей инструмента в работе мы использовали следующие:

1. Возможность выполнять тестовые воздействия, соответствующие переходам между состояниями в спецификации.

2. Возможность проверять соответствие свойств объектов реальной системы и эталонных свойств, содержащихся в спецификации. Из тех возможностей, которые доступны с помощью этого инструмента, используется проверка следующих свойств объектов:

- Наличие и состояние окон (заголовки, активность, доступность, статус).
- Наличие и состояние таких объектов, как PushButton, CheckBox, RadioButton, List, Tree и др. (текст, доступность, размер).
- Значение буфера обмена.
- Наличие в оперативной памяти запущенных процессов.
- Существование файлов.

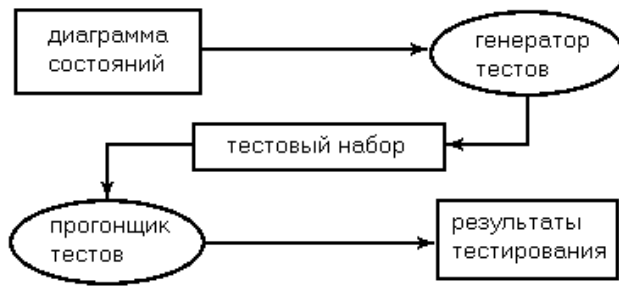


Рис. 2 Общая схема генерации и прогона тестов

Генератор строит по диаграмме состояний набор тестов. Условием окончания работы генератора является выполнение тестового покрытия. В данном случае в качестве покрытия было выбрано условие прохода по всем рёбрам графа состояний, то есть выполнения каждого доступного тестового воздействия из каждого достижимого состояния.

Автоматическая генерация тестов по диаграммам действий имеет следующие преимущества перед остальными подходами к тестированию графического интерфейса:

Генератор тестов есть программа (script), написанная на языке *‘Rational Rose Scripting language’* (расширение к языку *Summit BasicScriptLanguage*). В Rational Rose есть встроенный интерпретатор инструкций, написанных на этом языке, посредством которого можно обращаться ко всем объектам модели (диаграммы состояний).

- Спецификация автоматически интерпретируется (тем самым она проверяется и компилируется в набор тестов).
- Если какая-то функциональность системы изменилась, то диаграмму состояний достаточно изменить в соответствующем месте, и затем сгенерировать новый тестовый набор. Фактически, это снимает большую часть проблем, возникающих при организации регрессионного тестирования.
- Гарантия тестового покрытия. Эта гарантия даётся соответствующим алгоритмом обхода графа состояний.

В процессе построения обхода, генератор тестов компилирует набор тестов – инструкции на языке *SQABasic*. Эти инструкции есть чередование тестовых воздействий и оракула свойств объектов, соответствующих данному состоянию.

#### **Задание.**

1. Сформировать диаграмму вариантов использования для задачи лабораторной работы № 1.

2. Сгенерировать набор тестов.

3. Составить отчет по лабораторной работе.

**Отчет по лабораторной работе должен включать:**

1. Диаграмму вариантов использования.

Файл с тестовым набором.

### **Лабораторная работа «Разработка тестовых модулей проекта для тестирования отдельных модулей»**

**Цель работы.** Получение практических навыков использования средств автоматизации тестирования.

Теоретическая часть.

Для того чтобы продолжать тестирование, когда один тест не прошёл, в генератор тестов встроена возможность выбора – генерировать один большой тест или набор атомарных

тестов. Атомарный тест – тот, который не требует приведения системы в состояние, отличное от начального состояния.

В связи с наличием ограничения инструмента прогона тестов на тестовую длину, в тесты после каждой законченной инструкции вставляется строка разреза. Во время прогона по этим строкам осуществляется разрез теста в случае, если его длина превышает допустимое ограничение. После прохождения части теста до строки разреза продолжается выполнение теста с первой инструкции, следующей за строкой разреза. Нарезку и сам прогон тестов осуществляет прогонщик тестов.

В качестве прогонщика тестов мы используем Rational Robot, который выполняет сгенерированные наборы инструкций. В случае удачного выполнения всех инструкций выносится вердикт – тест прошёл. В противном случае, если на каком-то этапе выполнения теста, поведение системы не соответствует требованиям, Robot прекращает его выполнение, вынося соответствующий вердикт – тест не прошёл.

#### **Задание.**

1. Выполнить тестовый набор лабораторной работы № 2.
2. Проанализировать отчёт о прохождении тестов.
3. Составить отчет по лабораторной работе.

#### ***Отчет по лабораторной работе должен включать:***

1. Отчёт о прохождении тестов.
2. Анализ отчёта о прохождении тестов.

## **Лабораторная работа «Выполнение функционального тестирования»**

**Цель работы.** Получить практические навыки разработки тестов на основе внешней спецификации программы.

#### **Теоретические основы.**

Программа в случае тестирования с управлением по данным рассматривается как "черный ящик", и целью тестирования является выяснение обстоятельств, в которых поведение программы не соответствует спецификации. Различают следующие методы формирования тестовых наборов:

- эквивалентное разбиение;
- анализ граничных значений;
- анализ причинно-следственных связей;
- предположение об ошибке.

#### ***Эквивалентное разбиение.***

Область всех возможных наборов входных данных программы по каждому параметру разбивают на конечное число групп - классов эквивалентности. Наборы данных такого класса объединяют по принципу обнаружения одних и тех же ошибок. Для составления классов эквивалентности нужно перебрать ограничения, установленные для каждого входного значения в техническом задании или при уточнении спецификации. Каждое ограничение разбивают на две или более групп.

#### ***Граничные значения.***

Граничные значения - это значения на границах классов эквивалентности входных значений или около них.

#### ***Анализ причинно-следственных связей.***

Метод анализа причинно-следственных связей позволяет системно выбирать тесты, используя алгебру логики. Причиной называют отдельное входное условие или класс эквива-

лентности. Следствием - выходное условие или преобразование системы. Идея заключается в отнесении всех следствий к причинам, то есть в уточнении причинно-следственных связей.

***Предположение об ошибке.***

Метод основан на интуиции программиста с большим опытом работы. Составляется список, в котором перечисляются возможные ошибки или ситуации, в которых они могут появиться, а затем на основе списка составляются тесты.

**Задание.**

1. На основе внешней спецификации задачи Практического занятия №5 составить набор тестов на основе подхода «чёрного ящика».

2. Провести тестирование программы.

3. Оформить отчет по лабораторной работе.

***Отчет по лабораторной работе должен включать:***

1. Внешнюю спецификацию.

2. Алгоритм решения задачи.

3. Текст программы на языке программирования.

4. Набор тестов на основе подхода «чёрного ящика» для отладки программы.

## **Лабораторная работа «Тестирование интеграции»**

**Цель работы.** Получить практические навыки отладки программ с помощью отладчика среды программирования.

Теоретические основы.

Отладка — это процесс определения и устранения причин ошибок. Этим она отличается от тестирования, направленного на обнаружение ошибок. В некоторых проектах отладка занимает до 50% общего времени разработки. Многие программисты считают отладку самым трудным аспектом программирования.

Для сокращения времени отладки необходимо пользоваться научным подходом.

Классический научный подход включает следующие этапы:

1. Сбор данных при помощи повторяющихся экспериментов.

2. Формулирование гипотезы, объясняющей релевантные данные.

3. Разработка эксперимента, призванного подтвердить или опровергнуть гипотезу.

4. Подтверждение или опровержение гипотезы.

5. Повторение процесса в случае надобности.

Эффективный метод поиска дефектов при отладке с использованием научного подхода может быть описан следующими шагами:

1. Стабилизация ошибки.

2. Определение источника ошибки.

a. Сбор данных, приводящих к дефекту.

b. Анализ собранных данных и формулирование гипотезы, объясняющей дефект.

c. Определение способа подтверждения или опровержения гипотезы, основанного или на тестировании программы, или на изучении кода.

d. Подтверждение или опровержение гипотезы при помощи процедуры, определенной в п. 2(с).

3. Исправление дефекта.

4. Тестирование исправления.

5. Поиск похожих ошибок.

Способ подтверждения или опровержения гипотезы может быть одним из следующего списка:

1. сокращение подозрительной области кода;
2. проверка классов и методов, в которых дефекты обнаруживались ранее;
3. проверка кода, который изменялся недавно.

Отладка — это тот этап разработки программы, от которого зависит возможность ее выпуска. Конечно, лучше всего вообще избегать ошибок. Однако потратить время на улучшение навыков отладки все же стоит, потому что эффективность отладки, выполняемой лучшими и худшими программистами, различается минимум в 10 раз.

Систематичный подход к поиску и исправлению ошибок — неременное условие успешности отладки. Организуйте отладку так, чтобы каждый тест приближал вас к цели.

Используйте Научный Метод Отладки.

Прежде чем приступить к исправлению программы, поймите суть проблемы. Случайные предположения о причинах ошибок и случайные исправления только ухудшат программу.

Установите в настройках компилятора самый строгий уровень диагностики и устраняйте причины всех ошибок и предупреждений.

Инструменты отладки значительно облегчают разработку ПО. Найдите их и используйте. Большинство современных сред программирования (Delphi, C++ Builder, Visual Studio и т.д.) включают средства отладки, которые обеспечивают максимально эффективную отладку. Они позволяют:

- выполнять программу по шагам, причем как с заходом в подпрограммы, так и выполняя их целиком;
- предусматривать точки останова;
- выполнять программу до оператора, указанного курсором;
- отображать содержимое любых переменных при пошаговом выполнении;
- отслеживать поток сообщений и т.п.

#### **Задание.**

1. Составить в виде блок-схемы алгоритм решения задачи.
2. Создать программу решения задачи на любом алгоритмическом языке программирования.
3. Отладить программу с использованием инструментальных средств.
4. Составить отчет по лабораторной работе.

**Отчет по лабораторной работе должен включать:**

1. Алгоритм решения задачи.
2. Текст программы на языке программирования.
3. Набор тестов для отладки программы.

**Задача:** Имеется матрица размера  $N \times M$ . Определить в какой строке количество положительных элементов наибольшее.

#### **Контрольные вопросы.**

1. Что такое тестирование программы?
2. Что такое отладка программы?
3. Какие стадии тестирования выделяют при разработке программного обеспечения?
4. Какие различают подходы в формировании тестовых наборов?
5. В чем суть тестирования методом —покрытия операторов?
6. В чем суть тестирования методом —покрытия решений?
7. В чем суть тестирования методом —покрытия условий?
8. В чем суть тестирования методом —комбинаторного покрытия условий?
9. В чём суть метода эквивалентных разбиений?

10. В чём суть метода анализа граничных значений?

11. В чём суть метода анализа причинно-следственных связей?

## **Лабораторная работа «Документирование результатов тестирования»**

**Цель работы.** Получение практических навыков оформления протоколов тестирования и отладки программы.

Теоретические основы.

Тестирование – процесс выполнения программы на наборе тестов с целью выявления ошибок.

Обеспечить повторяемость процесса тестирования недостаточно – вы должны оценивать и проект, чтобы можно было точно сказать, улучшается он в результате изменений или ухудшается.

Вот некоторые категории данных, которые можно собирать с целью оценки проекта:

- административное описание дефекта (дата обнаружения, сотрудник, сообщивший о дефекте, номер сборки программы, дата исправления);
- полное описание проблемы;
- действия, предпринятые для воспроизведения проблемы;
- предложенные способы решения проблемы;
- родственные дефекты;
- тяжесть проблемы (например, критическая проблема, «неприятная» или косметическая);
- источник дефекта: выработка требований, проектирование, кодирование или тестирование;
- вид дефекта кодирования: ошибка занижения или завышения на 1, ошибка присваивания, недопустимый индекс массива, неправильный вызов метода и т. д.;
- классы и методы, измененные при исправлении дефекта;
- число строк, затронутых дефектом;
- время, ушедшее на нахождение дефекта;
- время, ушедшее на исправление дефекта.

Собирая эти данные, вы сможете подсчитывать некоторые показатели, позволяющие сделать вывод об изменении качества проекта:

- число дефектов в каждом классе; все числа целесообразно отсортировать в порядке от худшего класса к лучшему и, возможно, нормализовать по размеру класса;
- число дефектов в каждом методе, все числа целесообразно отсортировать в порядке от худшего метода к лучшему и, возможно, нормализовать по размеру метода;
- среднее время тестирования в расчете на один обнаруженный дефект;
- среднее число обнаруженных дефектов в расчете на один тест;
- среднее время программирования в расчете на один исправленный дефект;
- процент кода, покрытого тестами;
- число дефектов, относящихся к каждой категории тяжести.

Кроме протоколов тестирования уровня проекта, вы можете хранить и личные протоколы тестирования. Можете включать в них контрольные списки ошибок, которые вы допускаете чаще всего, и указывать время, затрачиваемое вами на написание кода, его тестирование и исправление ошибок.

**Задание.**

1. Выполнить тестирование программы, разработанной в лабораторной работе № 4.
2. Оформить протоколы тестирования.
3. Оформить отчет по лабораторной работе.

**Отчет по лабораторной работе должен включать:**

1. Внешнюю спецификацию.
2. Набор тестов.
3. Текст программы на языке программирования.
4. Протоколы тестирования программы.

**Типовой тест к дифференцированному зачету по темам МДК.02.02.****Тема «Отладка, тестирование и сопровождение программных продуктов»**

1. Под ошибкой подразумевается
  - 1) место в программе, где искажение проявляется или становится очевидным
  - 2) неправильность, погрешность или неумышленное искажение объекта или процесса
  - 3) место в программе, где возникают условия для появления искажений
  - 4) исправление выявленных искажений в процессе тестирования программы
1. Источником ошибок в программе может быть
  - 1) недостаточная квалификация специалиста
  - 2) сложность программы
  - 3) большой объем программы
  - 4) недостаточное знание заказчиком предметной области
2. Структурный подход к разработке программы является методом борьбы с...
  - 1) переводом программы
  - 2) неквалифицированностью специалиста
  - 3) взаимопониманием
  - 4) сложностью программы
3. Одним из признаков классификации ошибок является
  - 1) уровень сложности и устойчивости
  - 2) степень заикливания
  - 3) правильность описания программы
  - 4) возможность описания программы
4. Процесс отладки включает следующие подпроцессы:
  - 1) выявление ошибок, диагностика и локализация ошибок, исправление ошибок
  - 2) выявление ошибок и их локализация
  - 3) диагностика ошибок, исправление ошибок и повторное тестирование программы
  - 4) выявление ошибки, исправление ошибки
5. Отладка начинается с того момента как
  - 1) не выдается сообщение об ошибках
  - 2) не выдается сообщения о синтаксических ошибках
  - 3) программа полностью описана
  - 4) прописаны отдельные модули программы
6. Точка обнаружения – это...
  - 1) место в программе, где ошибка себя проявляет или становится очевидной
  - 2) неправильность, погрешность или неумышленное искажение объекта или процесса
  - 3) место в программе, где ошибку можно локализовать
  - 4) место в программе, где возникают условия для появления ошибки
7. Что может являться источником ошибки в программе?



- 1) перевод программы
- 2) недостаточная квалификация специалиста
- 3) модульное программирование
- 4) объектно-ориентированное программирование
8. Контроль (проверка, испытания) программы является методом борьбы с...
  - 1) переводом программы
  - 2) взаимопониманием
  - 3) сложностью программы
  - 4) описанием программы
9. Выделяют следующие виды ошибок программ:
  - 1) синтаксические, семантические, первичные
  - 2) ошибки анализа, общего и физического характера
  - 3) ошибки анализа, первичные и вторичные
  - 4) ошибки описания, определения функций и кодирования
10. Под отладкой понимается процесс
  - 1) нахождения и исправления ошибок
  - 2) позволяющий получить программу, которая функционирует с требующими характеристиками
  - 3) оптимизации программы
  - 4) тиражирования программы
11. Для тестирования программы используют
  - 1) простые тестовые данные
  - 2) просчитанные данные
  - 3) сложные данные
  - 4) произвольные данные
12. Точка происхождения – это...
  - 1) место в программе, где ошибка себя проявляет или становится очевидной
  - 2) неправильность, погрешность или неумышленное искажение объекта или процесса
  - 3) место в программе, где возникают условия для появления ошибки
  - 4) место в программе, где ошибку можно локализовать
13. Выберите возможные источники ошибки в программе.
  - 1) модульное программирование
  - 2) трудность во взаимопонимании между заказчиком и разработчиком
  - 3) сложность понимания языка программирования
  - 4) объектное описание программы
14. Переход на формальные стороны взаимодействия является методом борьбы с...
  - 1) переводом программы
  - 2) взаимопониманием
  - 3) сложностью программы
  - 4) пониманием языка программирования
15. Одним из признаков классификации ошибок является
  - 1) синтаксис и семантика
  - 2) степень заикливания
  - 3) первичные и побочные ошибки
  - 4) первостепенные и второстепенные ошибки
16. Отладка бывает:
  - 1) ручная и семантическая
  - 2) ручная и автоматизированная
  - 3) разрушающая и неразрушающая

- 4) разрушающая, семантическая, оптимизирующая
- 17. Тестирование – это...
  - 1) оптимизация программ
  - 2) действие, направленное на выявление ошибок
  - 3) регистрация программы
  - 4) исправление выявленных ошибок
- 18. Тестирование – это...
  - 1) процесс создания загрузочного файла программы
  - 2) запуск программы на выполнение
  - 3) процесс многократного выполнения программы с целью обнаружения максимального количества ошибок
  - 4) процесс нахождения и исправления ошибок
- 19. Тестовый набор данных должен включать
  - 1) входные, промежуточные и выходные данные
  - 2) входные и выходные данные
  - 3) все промежуточные результаты проверки тестов и конечный результат выполнения каждой функции
  - 4) входные, выходные данные и результаты проверки каждого условия
- 20. Тестирование бывает
  - 1) нисходящее, восходящее, промежуточное, завершенное
  - 2) структурное, функциональное, промежуточное, полное
  - 3) нисходящее, восходящее, структурное, полное
  - 4) нисходящее, восходящее, структурное, функциональное
- 21. Что известно при тестировании «черного ящика»?
  - 1) функции программы
  - 2) внутренняя структура программы
  - 3) работа каждой функции на всей области определения
  - 4) внутренние элементы программы и связи между ними
- 22. При тестировании «белого ящика» исследуется...
  - 1) функции программы
  - 2) внутренняя структура программы
  - 3) работа каждой функции на всей области определения
  - 4) внутренние элементы программы и связи между ними
- 23. К методам «белого ящика» относятся...
  - 1) метод покрытия решений, метод граничных решений, метод функциональных диаграмм, метод покрытия условий
  - 2) метод эквивалентных разбиений, метод функциональных диаграмм, анализ граничных решений
  - 3) метод покрытия условий, метод покрытия операторов, метод покрытия решений, анализ граничных решений
  - 4) метод покрытия условий, метод покрытия операторов, метод покрытия решений, метод покрытия решений и условий
- 24. Метод эквивалентных разбиений основан на...
  - 1) разработке такого числа эквивалентных тестов, достаточного для того, что бы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз
  - 2) разбиении входной области программы на классы по определенным признакам
  - 3) разработке достаточного количества тестов, чтобы каждое решение на этих тестах выполнялось по крайней мере один раз
  - 4) выполнении каждого оператора хотя бы один раз

25. Метод покрытия условий основан на...

- 1) разработке такого числа эквивалентных тестов, достаточного для того, чтобы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз
- 2) разбиении входной области программы на классы по определенным признакам
- 3) разработке достаточного количества тестов, чтобы возможные результаты каждого условия в решении выполнялось по крайней мере один раз
- 4) выполнении каждого оператора хотя бы один раз

26. Тестирование «черного ящика» выполняется

- 1) на ранних этапах разработки программы
- 2) когда разработан весь программный продукт и протестированы отдельные его модули
- 3) на поздних стадиях тестирования программы
- 4) на ранних стадиях тестирования программы

27. Техника «черного ящика» ориентирована на...

- 1) выявление класса ошибок
- 2) выявление отдельных ошибок
- 3) сокращение количества тестовых вариантов
- 4) увеличение количества тестовых наборов

28. Тестирование включает в себя ...

- 1) создание текстового, загрузочного файла и их проверка
- 2) разработка тестов и непосредственное тестирование по ним
- 3) проверка разработанного набора тестов на исполняемом файле
- 4) составление алгоритма решения задачи, текста программы, набора тестовых данных и их проверка

29. Чему равна вероятность наличия необнаруженных ошибок в какой-то части программы?

- 1) обратно пропорциональна числу ошибок, обнаруженных в программе
- 2) количеству обнаруженных в программе ошибок
- 3) пропорциональна числу ошибок, обнаруженных в программе
- 4)  $1/3$  числу обнаруженных ошибок

30. Что известно при тестировании «белого ящика»?

- 1) функции программы
- 2) внутренняя структура программы
- 3) работа каждой функции на всей области определения
- 4) внутренние элементы программы и связи между ними

31. При тестировании «черного ящика» исследуется...

- 1) функции программы
- 2) внутренняя структура программы
- 3) работа каждой функции на всей области определения
- 4) внутренние элементы программы и связи между ними

32. К методам «черного ящика» относятся...

- 1) метод покрытия решений, метод граничных решений, метод функциональных диаграмм, метод покрытия условий
- 2) метод эквивалентных разбиений, метод функциональных диаграмм, анализ граничных решений
- 3) метод покрытия условий, метод покрытия операторов, метод покрытия решений, анализ граничных решений
- 4) метод покрытия условий, метод покрытия операторов, метод покрытия решений, метод покрытия решений и условий

33. Метод покрытия операторов при тестировании программ основан на...

- 1) разработке такого числа эквивалентных тестов, достаточного для того, что бы все возможные результаты каждого условия в решении выполнялись по крайней мере один раз
  - 2) разбиении входной области программы на классы по определенным признакам
  - 3) разработке достаточного количества тестов, чтобы каждое решение на этих тестах выполнялось по крайней мере один раз
  - 4) выполнении каждого оператора хотя бы один раз
34. Граничные условия – это
- 1) условия, ситуация, возникающая непосредственно на границе выше или ниже границ входных или выходных элементов класса эквивалентности
  - 2) ситуация, возникающая непосредственно на промежуточных элементах класса эквивалентности
  - 3) условия, ситуация, возникающие внутри программы, когда выполнены все тестовые наборы
35. При тестировании программ методами «черного ящика» необходимо разрабатывать набор тестов, который...
- 1) показывает нормальное функционирование программы
  - 2) выявляет все ошибки программы и по ним позволяет оптимизировать программу
  - 3) показывает нормальное и аномальное функционирование программы
36. Тестирование «белого ящика» выполняется
- 1) на ранних этапах разработки программы
  - 2) когда разработан весь программный продукт и протестированы отдельные его модули
  - 3) на поздних стадиях тестирования программы
  - 4) на ранних стадиях тестирования программы
37. Тестирование «черного ящика» обеспечивает поиск следующих категорий ошибок:
- 1) ошибок во внутренних структурах данных
  - 2) ошибок интерфейса
  - 3) ошибок во внешних структурах данных
  - 4) ошибок в циклах и ветвлениях
  - 5) ошибок характеристик
38. К программным средствам защиты программного продукта не относят....
- 1) криптографическую защиту
  - 2) ограничение доступа к программному продукту
  - 3) патентную защиту
  - 4) нестандартное форматирование диска, на котором находится программный продукт
39. Лицензирование программного продукта относится к...
- 1) правовой защите ПП
  - 2) программной защите ПП
  - 3) технической защите ПП
  - 4) физической защите ПП
40. 41. Каким знаком обозначается авторское право на программный продукт?
- 1) ©
  - 2) тм
  - 3) ®
41. Каким знаком обозначается регистрация права на программный продукт?
- 1) ©
  - 2) тм
  - 3) ®
42. Этап Эволюции при сопровождении программного продукта предполагает...

- 1) выявление и устранение обнаруженных ошибок, тиражирование, контроль за распространением версии, введение новых функций программы и т.д
  - 2) внесение изменения в программу в ответ на изменившиеся условия
  - 3) использование всех возможных и невозможных способов для поддержания жизни в старой и распадающейся на части программной системе
  - 4) проектирование программного продукта, тестирование, тиражирование и утилизацию
43. Этап Сохранение при сопровождении программного продукта предполагает...
- 1) выявление и устранение обнаруженных ошибок, тиражирование, контроль за распространением версии, введение новых функций программы и т.д
  - 2) внесение изменения в программу в ответ на изменившиеся условия
  - 3) использование всех возможных и невозможных способов для поддержания жизни в старой и распадающейся на части программной системе
  - 4) проектирование программного продукта, тестирование, тиражирование и утилизацию
44. Этап Чистое сопровождение при сопровождении программного продукта предполагает...
- 1) выявление и устранение обнаруженных ошибок, тиражирование, контроль за распространением версии, введение новых функций программы и т.д
  - 2) внесение изменения в программу в ответ на изменившиеся условия
  - 3) использование всех возможных и невозможных способов для поддержания жизни в старой и распадающейся на части программной системе
  - 4) проектирование программного продукта, тестирование, тиражирование и утилизацию

#### **Тема «Коллективная разработка программных средств»**

1. Существует две основные модели организации коллектива при разработке ПО:
  - 1) иерархическая модель и модель группы
  - 2) структурная и объектная модель
  - 3) иерархическая и объектная модель
  - 4) модель группы и сетевая модель
2. Какая модель коллективной разработки программного продукта определяет структуру коллектива с точки зрения отдела кадров?
  - 1) модель группы
  - 2) иерархическая модель
  - 3) структурная модель
  - 4) сетевая модель
3. Какая модель коллективной разработки программного продукта не определяет структуру коллектива с точки зрения отдела кадров?
  - 1) модель группы
  - 2) иерархическая модель
  - 3) структурная модель
  - 4) сетевая модель
4. Основными недостатками иерархической модели коллективной разработки программных продуктов является:
  - 1) несогласованное представление о разных сторонах проекта
  - 2) нехватка информации
  - 3) разрозненная связь с внешними источниками информации
  - 4) сложность расстановки приоритетов
5. Основными недостатками групповой модели коллективной разработки программных продуктов является:
  - 1) несогласованное представление о разных сторонах проекта

- 2) нехватка информации
- 3) разрозненная связь с внешними источниками информации
- 4) сложность расстановки приоритетов
6. Для скрытия недостатков иерархической модели коллективной разработки программных продуктов предусматривают
  - 1) сплочение коллектива путем приобретения большого числа заказов
  - 2) распределение обязанностей руководителя между членами коллектива
  - 3) объединение обязанностей руководителя и отдельных членов коллектива
  - 4) определение целей проекта и распределение обязанностей в соответствии с целями
7. В чем заключается задача модели проектной группы при коллективной разработке программного продукта?
  - 1) сплочение коллектива путем приобретения большого числа заказов
  - 2) распределение обязанностей руководителя между членами коллектива
  - 3) объединение обязанностей руководителя и отдельных членов коллектива
  - 4) определение целей проекта и распределение обязанностей между членами группы
8. Основной целью менеджера продукта при коллективной разработке программного продукта является:
  - 1) удовлетворение требований заказчика
  - 2) соблюдение ограничений проекта
  - 3) соответствие спецификациям
  - 4) выпуск программного продукта только после выявления и устранения проблем
9. Основной целью менеджера программы при коллективной разработке программного продукта является:
  - 1) удовлетворение требований заказчика
  - 2) соблюдение ограничений проекта
  - 3) соответствие спецификациям
  - 4) выпуск только после выявления и устранения проблем
10. Основной целью разработчика при коллективной разработке программного продукта является:
  - 1) удовлетворение требований заказчика
  - 2) соблюдение ограничений проекта
  - 3) соответствие спецификациям
  - 4) выпуск программного продукта только после выявления и устранения проблем
11. Основной целью тестера при коллективной разработке программного продукта является:
  - 1) удовлетворение требований заказчика
  - 2) соблюдение ограничений проекта
  - 3) соответствие спецификациям
  - 4) выпуск программного продукта только после выявления и устранения проблем
12. Основной целью инструктора при коллективной разработке программного продукта является:
  - 1) удовлетворение требований заказчика
  - 2) соблюдение ограничений проекта
  - 3) повышение эффективности труда пользователя
  - 4) выпуск программного продукта только после выявления и устранения проблем
13. Основной целью инструктора при коллективной разработке программного продукта является:
  - 1) удовлетворение требований заказчика
  - 2) простота развертывания и постоянное сопровождение программного продукта

- 3) повышение эффективности труда пользователя
- 4) выпуск программного продукта только после выявления и устранения проблем
14. Главная задача менеджера продукта при коллективной разработке программного продукта заключается...
  - 1) в формировании общего представления о поставленной задаче и о том, как ее решать
  - 2) в ведении процесса разработки с учетом всех ограничений
  - 3) в испытании продукта в реальных условиях
  - 4) в повышении эффективности труда пользователей
15. Главная задача тестера при коллективной разработке программного продукта заключается...
  - 1) в формировании общего представления о поставленной задаче и о том, как ее решать
  - 2) в ведении процесса разработки с учетом всех ограничений
  - 3) в испытании продукта в реальных условиях
  - 4) в повышении эффективности труда пользователей
16. Главная задача менеджера продукта при коллективной разработке программного продукта заключается...
  - 1) в формировании общего представления о поставленной задаче и о том, как ее решать
  - 2) в ведении процесса разработки с учетом всех ограничений
  - 3) в испытании продукта в реальных условиях
  - 4) в повышении эффективности труда пользователей
17. Главная задача инструктора при коллективной разработке программного продукта состоит...
  - 1) в формировании общего представления о поставленной задаче и о том, как ее решать
  - 2) в ведении процесса разработки с учетом всех ограничений
  - 3) в испытании продукта в реальных условиях
  - 4) в повышении эффективности труда пользователей
18. Главная задача логистика при коллективной разработке программного продукта состоит...
  - 1) в формировании общего представления о поставленной задаче и о том, как ее решать
  - 2) в проверке, чтобы все серверы развертывания и рабочие станции пользователей удовлетворяли указанным требованиям
  - 3) в испытании продукта в реальных условиях
  - 4) в повышении эффективности труда пользователей
19. Кто из членов группы при коллективной разработке программных продуктов составляет график работ?
  - 1) менеджер продукта
  - 2) менеджер программы
  - 3) логистик
  - 4) инструктор
20. Кто из членов группы при коллективной разработке программных продуктов выполняет проектирование архитектуры программного продукта?
  - 1) менеджер продукта
  - 2) разработчик
  - 3) логистик
  - 4) инструктор
21. Кто из членов группы при коллективной разработке программных продуктов разрабатывает стратегию, планы, графики и сценарии тестирования?
  - 1) менеджер продукта
  - 2) разработчик

3) логистик

4) тестер

22. Кто из членов группы при коллективной разработке программных продуктов составляет документацию, определяет требования к резервному копированию данных и разрабатывает план восстановления на случай отказа систем?

1) менеджер продукта

2) разработчик

3) логистик

4) тестер

23. Кто из членов группы при коллективной разработке программных продуктов участвует в создании пользовательского интерфейса, сокращая тем самым затраты на сопровождение продукта и поддержку пользователей?

1) менеджер продукта

2) разработчик

3) инструктор

4) тестер

24. К недостатками иерархической модели коллективной разработки программных продуктов можно отнести:

1) несогласованное представление о разных сторонах проекта

2) невозможностью учесть все особенности проекта

3) разрозненная связь с внешними источниками информации

4) отсутствием полноценной связи между всеми участниками проекта, так как вся информация идет в одном направлении — вверх по иерархии, к главному менеджеру

25. К недостаткам иерархической модели коллективной разработки программных продуктов можно отнести:

1) несогласованность личных планов членов группы

2) невозможностью учесть все особенности проекта

3) отсутствие опыта, снижающее эффективность коллективной работы

4) отсутствием полноценной связи между всеми участниками проекта, так как вся информация идет в одном направлении — вверх по иерархии, к главному менеджеру

26. Какие задачи необходимо решить, чтобы проект считался удачным?

1) удовлетворить требования заказчика

2) соблюсти ограничения

3) спроектировать систему по объектно-ориентированному методу

4) выполнить спецификации, основанные на требованиях пользователей

5) выпустить продукт только после выявления и устранения всех проблем

6) выполнить программный продукт с учетом ситуации на рынке программ

7) гарантировать простоту развертывания и управления