

**Министерство сельского хозяйства Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования «Дагестанский государственный аграрный  
университет имени М.М. Джамбулатова»**

**Аграрно-экономический техникум**



**Методические указания  
по выполнению практических работ**

**ПМ.02. ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ  
МОДУЛЕЙ  
МДК 02.02 ИНТЕГРАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ  
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**09.02.07 Информационные системы и программирование**

**Махачкала 2023**

Методические указания по выполнению практических работ по МДК 02.02 Инструментальные средства разработки программного обеспечения работ разработаны в соответствии с рабочей программой ПМ.02. Осуществление интеграции программных модулей по специальности 09.02.07 Информационные системы и программирование.

**СОГЛАСОВАНО:**



Директор АЭТ

подпись

Магомедов Д.А.

**Одобрено на заседании ПЦК общепрофессиональных,  
специальных дисциплин "10" марта 2023 г., протокол № 7.**



Председатель ПЦК

(подпись)

Касимовская О.О.  
(инициалы, фамилия)

**СОГЛАСОВАНО:**

Директор Компании Color- IT, Интернет решения



Салихов А.Б.  
Ф.И.О.

## СОДЕРЖАНИЕ

Пояснительная записка .....	4
Порядок выполнения практической работы .....	5
Рекомендации по оформлению практической работы .....	5
Критерии оценки практической работы .....	5
Перечень практических работ .....	6
Лабораторная работа №11 «Разработка структуры проекта» .....	7
Лабораторная работа №12 «Разработка модульной структуры проекта (диаграммы модулей)» .....	9
Лабораторная работа №13 «Разработка перечня артефактов и протоколов проекта» .....	11
Лабораторная работа №14 «Настройка работы системы контроля версий (типов импортируемых файлов, путей, фильтров и др. параметров импорта в репозиторий)» ...	13
Лабораторная работа №15 «Разработка и интеграция модулей проекта (командная работа)» .....	15
Лабораторная работа №16 «Отладка отдельных модулей программного проекта» .....	17
Лабораторная работа №17 «Организация обработки исключений» .....	20
Лабораторная работа №18 «Применение отладочных классов в проекте» .....	21
Лабораторная работа №19 «Отладка проекта» .....	21
Лабораторная работа №20 «Инспекция кода модулей проекта» .....	23
Лабораторная работа №21 «Тестирование интерфейса пользователя средствами инструментальной среды разработки» .....	29
Лабораторная работа №22 «Разработка тестовых модулей проекта для тестирования отдельных модулей» .....	31
Лабораторная работа №23 «Выполнение функционального тестирования» .....	32
Лабораторная работа №24 «Тестирование интеграции» .....	33
Лабораторная работа №25 «Документирование результатов тестирования» .....	35

### Пояснительная записка

Методические рекомендации по выполнению лабораторных и практических работ по МДК 02.02 Инструментальные средства разработки программного обеспечения разработаны в соответствии с рабочей программой профессионального модуля и предназначены для приобретения необходимых практических навыков и закрепления теоретических знаний, полученных обучающимися при изучении профессионального модуля, обобщения и систематизации знаний перед экзаменом.

Методические рекомендации предназначены для обучающихся специальности 09.02.07 Информационные системы и программирование.

МДК 02.02 Инструментальные средства разработки программного обеспечения относится к профессиональному циклу, изучается на 2 курсе и при его изучении отводится значительное место выполнению практических работ.

Освоение содержания МДК 02.02 Инструментальные средства разработки программного обеспечения во время выполнения практических работ обеспечивает достижение обучающимися следующих **результатов**:

Код	Наименование общих компетенций
ОК 1.	Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам
ОК 2.	Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.
ОК 3	Планировать и реализовывать собственное профессиональное и личностное развитие.
ОК 4	Планировать и реализовывать собственное профессиональное и личностное развитие.
ОК 5	Планировать и реализовывать собственное профессиональное и личностное развитие.
ОК 6	Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей
ОК 7	Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.
ОК 8	Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности
ОК 9	Использовать информационные технологии в профессиональной деятельности.
ОК 10	Пользоваться профессиональной документацией на государственном и иностранном языке
ОК 11	Планировать предпринимательскую деятельность в профессиональной сфере
ВД 2	Осуществление интеграции программных модулей
ПК 2.2.	Выполнять интеграцию модулей в программное обеспечение
ПК 2.3	Выполнять отладку программного модуля с использованием специализированных программных средств
ПК 2.5.	Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования

В результате освоения профессионального модуля обучающийся должен:

Иметь практический опыт	использования инструментальных средств обработки информации; участия в разработке технического задания; формирования отчетной документации по результатам работ; использования стандартов при оформлении программной документации; программирования в соответствии с требованиями технического задания; использования критериев оценки качества и надежности функционирования информационных систем; применения методики тестирования
-------------------------	---

	разрабатываемых приложений; управления процессом разработки приложений с использованием инструментальных средств;
Уметь	осуществлять математическую и информационную постановку задач по обработке информации, использовать алгоритмы обработки информации для различных приложений; уметь решать прикладные вопросы интеллектуальных систем с использованием, статических экспертных систем, экспертных систем реального времени; использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ, разрабатывать графический интерфейс приложения; создавать проект по разработке приложения и формулировать его задачи, выполнять управление проектом с использованием инструментальных средств;
Знать	основные виды и процедуры обработки информации, модели и методы решения задач обработки информации (генерация отчетов, поддержка принятия решений, анализ данных, искусственный интеллект, обработка изображений); сервисно – ориентированные архитектуры, CRM-системы, ERP-системы; объектно-ориентированное программирование; спецификации языка, создание графического пользовательского интерфейса (GUI), файловый ввод-вывод, создание сетевого сервера и сетевого клиента; платформы для создания, исполнения и управления информационной системой; основные процессы управления проектом разработки.

В соответствии с рабочей программой ПМ.02. Осуществление интеграции программных модулей практические работы по МДК 02.02 Инструментальные средства разработки программного обеспечения разбиты на два семестра – третий и четвертый семестры. Целесообразность данной группировки обусловлена необходимостью обобщения и систематизации знаний перед экзаменом.

Рабочая программа профессионального модуля предусматривает проведение практических работ по МДК 02.02 Инструментальные средства разработки программного обеспечения в объеме 30 часов.

#### **Порядок выполнения практической работы**

- записать название работы, ее цель в тетрадь;
- выполнить основные задания в соответствии с ходом работы;
- выполнить индивидуальные задания.

#### **Рекомендации по оформлению практической работы**

Задания выполняются обучающимися по шагам. Необходимо строго придерживаться порядка действий, описанного в практической работе

Результаты выполнения практических работ необходимо сохранять в своей папке на компьютере или USB – накопителе.

В случае пропуска занятий обучающийся осваивает материал самостоятельно в свободное от занятий время и сдает практическую работу с пояснениями о выполнении.

#### **Критерии оценки практической работы**

- наличие цели выполняемой работы, выполнение более половины основных заданий (удовлетворительно);
- наличие цели выполняемой работы, выполнение всех основных и более половины дополнительных заданий (хорошо);
- наличие цели выполняемой работы, выполнение всех основных и индивидуальных заданий (отлично).

### Перечень практических работ

№	Наименование разделов и тем профессионального модуля (ПМ)	Наименование лабораторных работ и практических занятий	Объем часов
	Раздел 2. Средства разработки программного обеспечения		
	МДК 02.02 Инструментальные средства разработки программного обеспечения		
1	Тема 2.2.1 Современные технологии и инструменты интеграции	Лабораторная работа №11 «Разработка структуры проекта»	2
2		Лабораторная работа №12 «Разработка модульной структуры проекта (диаграммы модулей)»	2
3		Лабораторная работа №13 «Разработка перечня артефактов и протоколов проекта»	2
4		Лабораторная работа №14 «Настройка работы системы контроля версий (типов импортируемых файлов, путей, фильтров и др. параметров импорта в репозиторий)»	2
5		Лабораторная работа №15 «Разработка и интеграция модулей проекта (командная работа)»	2
6		Лабораторная работа №16 «Отладка отдельных модулей программного проекта»	2
7		Лабораторная работа №17 «Организация обработки исключений»	2
8	Тема 2.2.2 Инструментарий тестирования и анализа качества программных средств	Лабораторная работа №18 «Применение отладочных классов в проекте»	2
9		Лабораторная работа №19 «Отладка проекта»	2
10		Лабораторная работа №20 «Инспекция кода модулей проекта»	2
11		Лабораторная работа №21 «Тестирование интерфейса пользователя средствами инструментальной среды разработки»	2
12		Лабораторная работа №22 «Разработка тестовых модулей проекта для тестирования отдельных модулей»	2
13		Лабораторная работа №23 «Выполнение функционального тестирования»	2
14		Лабораторная работа №24 «Тестирование интеграции»	2
15		Лабораторная работа №25 «Документирование результатов тестирования»	2
Итого			90

## Лабораторная работа №11 «Разработка структуры проекта»

### Цель работы:

Формирование навыков постановки задачи и разработки технического задания на программный продукт.

### Задание

1. Выбрать вариант задания на проектирование и разработку учебной программы.  
2. В соответствии с вариантом выполнить разработку технического задания, которое должно включать:

- о введение;
  - о основание для разработки;
  - о назначение;
  - о требования к программе и программному продукту;
  - о требования к программной документации.
3. Оформить отчет. Содержание отчета:
- о тема лабораторной работы
  - о цель лабораторной работы
  - о ответы на контрольные вопросы
  - о задание на лабораторную работу
  - о разработанное техническое задание
  - о выводы по проделанной работе.

### Варианты заданий

1. Ввести вещественную матрицу размерности  $n * m$  построчно, а вывести по столбцам.
2. Выяснить сколько положительных элементов содержит матрица размерности  $n * m$ , если  $a_{ij} = \sin(i+j/2)$ .
3. Дана квадратная вещественная матрица размерности  $n$ . Является ли матрица симметричной относительно главной диагонали.
4. Дана квадратная вещественная матрица размерности  $n$ . Транспонировать матрицу.
5. Дана квадратная вещественная матрица размерности  $n$ . Сравнить сумму элементов матрицы на главной и побочной диагоналях.
6. Дана квадратная вещественная матрица размерности  $n$ . Найти количество нулевых элементов, стоящих:
  - выше главной диагонали;
  - ниже главной диагонали;
  - выше и ниже побочной.
7. Дана вещественная матрица размерности  $n * m$ . По матрице получить логический вектор, присвоив его  $k$ -ому элементу значение `True`, если выполнено указанное условие и значение `False` иначе:
  - все элементы  $k$  столбца нулевые;
  - элементы  $k$  строки матрицы упорядочены по убыванию;
  - $k$  строка массива симметрична.
8. Дана вещественная матрица размерности  $n * m$ . Сформировать вектор  $b$ , в котором элементы вычисляются как:
  - произведение элементов соответствующих строк;
  - среднее арифметическое соответствующих столбцов;
  - разность наибольших и наименьших элементов соответствующих строк;
  - значения первых отрицательных элементов в столбце.
9. Дана вещественная матрица размерности  $n * m$ . Вывести номера столбцов, содержащих только отрицательные элементы.
10. Дана вещественная матрица размерности  $n * m$ . Вывести номера строк, содержащих больше положительных элементов, чем отрицательных.
11. Дана вещественная матрица размерности  $n * m$ . Найти общую сумму элементов только тех столбцов, которые имеют хотя бы один нулевой элемент.
12. Дана вещественная матрица размерности  $n * m$ . Поменять местами строки с максимальным и минимальным элементами.
13. Дана вещественная матрица размерности  $n * m$ . Удалить  $k$  столбец матрицы.
14. Дана вещественная квадратная матрица размерности  $n$ . Поменять местами элементы главной и побочной диагоналей матрицы:

по строкам;  
по столбцам.

15. Дана вещественная матрица размерности  $m * n$ . Упорядочить элементы каждой четной строки по возрастанию.
16. Дана вещественная матрица размерности  $m * n$ . Расположить все элементы матрицы по убыванию. Обход матрицы осуществлять по строкам.
17. Дана вещественная матрица размерности  $m * n$ . Определить индексы первого нулевого элемента матрицы. Обход матрицы осуществлять по столбцам.
18. Известно положение двух ферзей на шахматной доске. Бьют ли они друг друга?

#### **Контрольные вопросы**

1. Перечислите этапы разработки программных продуктов.
2. Для чего необходимо техническое задание?
3. Кто занимается разработкой технического задания?
4. Какие пункты включает техническое задание?

#### **Содержание отчета**

- 1) Титульный лист.
- 2) Наименование и цель работы.
- 3) Краткое теоретическое описание.
- 4) Задание на лабораторную работу.
- 5) Листинг программы.
- 6) Результаты выполнения программы.



## **Лабораторная работа №12 «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 обзорным, 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 недоговорок.

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 №13 «Разработка перечня артефактов и протоколов проекта» Системный анализ и пути решения задачи

При разработке ПС человек имеет дело с системами. Под *системой* будем понимать совокупность взаимодействующих (находящихся в отношениях) друг с другом элементов. ПС можно рассматривать как пример системы. Логически связанный набор программ является другим примером системы. Любая отдельная программа также является системой. Понять систему — значит осмысленно перебрать все пути взаимодействия между ее элементами.

Целью системного анализа в наиболее общем виде является описание и исследование систем, определение путей и методов разработки ПО. Система характеризуется структурой и поведением. Применительно к разработке ПО системный анализ представляет собой анализ существующей структуры отношений в рамках конкретной предметной области, выявление роли и места будущей программной системы, ее основных функций и свойств. В этой связи системный анализ также можно назвать *внешним проектированием*.

Этап системного анализа состоит из следующих трех стадий:

1. обоснование необходимости разработки программы;
2. научно-исследовательские работы (НИР);
3. разработка и утверждение технического задания.

На первой стадии выполняются постановка задачи, сбор исходных материалов, Выбор и обоснование критериев эффективности и качества разрабатываемой программы, обоснование необходимости проведения научно-исследовательских работ.

На стадии научно-исследовательских работ решаются следующие задачи: определяется структура входных и выходных данных, осуществляется предварительный выбор методов решения задач, обосновывается целесообразность применения ранее разработанных программ, определяются требования к техническим средствам, обосновывается принципиальная возможность решения поставленной задачи.

На стадии разработки и утверждения технического задания определяются требования к программе, разрабатываются технико-экономического обоснования разработки программ, определяются стадии, этапы и сроки разработки программы и документации на нее, согласовывается и утверждается *техническое задание*.

Результат системного анализа — **спецификация** (техническое задание) как самостоятельный документ имеет очень важное значение. Этот документ является формальным соглашением между заказчиком продукта и его разработчиками.

В настоящее время можно выделить пять основных подходов к организации процесса создания и использования программного обеспечения.

1. *Водопадный подход*. При таком подходе разработка ПС состоит из цепочки этапов. На каждом этапе создаются документы, используемые на последующем этапе. В исходном документе фиксируются требования к ПС. В конце этой цепочки создаются программы, включаемые в ПС.
2. *Исследовательское программирование*. Этот подход предполагает быструю (насколько это возможно) реализацию рабочих версий программ ПС, выполняющих лишь в первом приближении требуемые функции. После экспериментального применения реализованных программ производится их модификация с целью сделать их более полезными для пользователей. Этот процесс повторяется до тех пор, пока ПС не будет достаточно приемлемо для пользователей. Такой подход применялся на ранних этапах развития программирования, когда технологии программирования не придавали большого значения (использовалась интуитивная технология). В настоящее время этот подход применяется для разработки таких ПС, для которых пользователи не могут точно сформулировать требования (например, для разработки систем искусственного интеллекта).
3. *Прототипирование*. Этот подход моделирует начальную фазу исследовательского программирования вплоть до создания рабочих версий программ, предназначенных для проведения экспериментов с целью установить требования к ПС. В дальнейшем должна последовать разработка ПС по установленным требованиям в рамках какого-либо другого подхода (например, водопадного).
4. *Формальные преобразования*. Этот подход включает разработку формальных спецификаций ПС и превращение их в программы путем корректных преобразований.

На этом подходе базируется компьютерная технология (CASE-технология) разработки ПС.

5. *Сборочное программирование.* Этот подход предполагает, что ПС конструируется, главным образом, из компонентов, которые уже существуют. Должно быть некоторое хранилище (библиотека) таких компонентов, каждая из которых может многократно использоваться в разных ПС. Такие компоненты называются *повторно используемыми(reusable)*. Процесс разработки ПС при данном подходе состоит скорее из сборки программ из компонентов, чем из их программирования.

#### **Задание**

1. В соответствии с подготовленным техническим заданием выполнить разработку спецификаций на программный продукт, которые должны включать:
  - о спецификации процессов;
  - о словарь терминов;
  - о диаграммы переходов состояний;
  - о диаграммы потоков с детализацией.
2. Оформить отчет. Содержание отчета:
  - о тема лабораторной работы;
  - о цель лабораторной работы;
  - о ответы на контрольные вопросы;
  - о задание на лабораторную работу;
  - о разработанные спецификации процессов;
  - о словарь терминов;
  - о диаграммы переходов состояний;
  - о диаграммы потоков с детализацией;
  - о выводы по проделанной работе.

#### **Контрольные вопросы**

1. Для чего разрабатываются спецификации на программный продукт?
2. Что должны включать спецификации на программный продукт?
3. Что должна содержать спецификация процессов
4. Что такое словарь терминов и для чего он используется?
5. Что такое диаграмма переходов состояний и для чего ее используют?
6. Что такое диаграмма потоков и для чего ее используют?

## **Лабораторная работа №14 «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 (CC 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)

Ссылки на учебные репозитории:

- Git
  - [https://github.com/irgups/project\\_2015\\_01.git](https://github.com/irgups/project_2015_01.git)
  - [https://github.com/irgups/project\\_2015\\_02.git](https://github.com/irgups/project_2015_02.git)
- Subversion
  - [https://github.com/irgups/project\\_2015\\_01](https://github.com/irgups/project_2015_01)
  - [https://github.com/irgups/project\\_2015\\_02](https://github.com/irgups/project_2015_02)

## Лабораторная работа №15 «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 10x10 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 10x10 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 знакопеременную 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 10x10 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 10x10 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 10x10 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 №16 «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. При этом 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 исполняется 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 повторяющих 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 выявления 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	$Y(X)$ ,		Вид	функции	$Y(X)$ ,
--	-----	---------	----------	--	-----	---------	----------

№ п/п	характерные точки	№ п/п	характерные точки
	Локальный минимум функции $x^2 + 0.5 - \sin(3x)$	6	Точка (точки), в которой функция $\left  \frac{2x^3 - 3}{x^2 + 1} \right $ равна 10.
	Точка (точки), для которой $\sqrt{x^2 + 5} - 1 = 5$ .	7	Локальный минимум функции $3(x + 4)^2 - \sin(x + 15)$
	Пересечение графиков функций $\sqrt{x^2 + 1}(3 + x)$ и $\frac{5}{x^2 - 7}$ .	8	Точка (точки), в которой функция $\frac{2x^2}{\sin(x - 1)} - 1$ равна -500.
	Все нули функции $x^2 - \sin(3x)$ .	9	Локальный максимум функции $\frac{2x + 15}{3 + x^2}$
	Локальный минимум функции $\frac{5}{3x - x^2 - 3}$	0	Пересечение графиков функций $(x^2 + 1)\sin(3 + x)$ и $\frac{x + 5}{x^2 + 1}$ .
	Точка (точки), в которой функция $x^2 - e^x$ равна -10.	1	Локальный минимум функции $x^2 - 3x + 15$
	Пересечение графиков функций $100\sin(1 + x)$ и $x^3 + 5$ .	2	Все нули функции $\sqrt{x^2 + 0.5} - \sin(3x)$ .
	Локальный минимум функции $\sqrt{e^x} - x$	3	Локальный максимум функции $200x - e^x$
	Все нули функции $\frac{x^2 - 15}{\ln x^2 + 3x - 7}$ .	4	Все нули функции $x - e^x \cos(x)$ .
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 №17 «Организация обработки исключений»**

## **Лабораторная работа №18 «Применение отладочных классов в проекте»**

### **Лабораторная работа №19 «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.

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

## Лабораторная работа №20 «Инспекция кода модулей проекта»

**Цель работы** — получить практические навыки разработки модулей программной системы и интеграции этих модулей.

**Теоретические сведения** Термин «интеграция» относится к такой операции в процессе разработки ПО, при которой вы объединяете отдельные программные компоненты в единую систему. В небольших проектах интеграция может занять одно утро и заключаться в объединении горстки классов. В больших — могут потребоваться недели или месяцы, чтобы связать воедино весь набор программ. Независимо от размера задач в них применяются одни и те же принципы.

Тема интеграции тесно переплетается с вопросом последовательности конструирования. Порядок, в котором вы создаете классы или компоненты, влияет на порядок их интеграции: вы не можете интегрировать то, что еще не было создано. Последовательности интеграции и конструирования имеют большое значение.

Поскольку интеграция выполняется после того, как разработчик завершил модульное тестирование, и одновременно с системным тестированием, ее иногда считают операцией, относящейся к тестированию. Однако она достаточно сложна, и поэтому ее следует рассматривать как независимый вид деятельности.

Аккуратная интеграция обеспечивает:

- упрощенную диагностику дефектов;
- меньшее число ошибок;
- меньшее количество «лесов»;
- раннее создание первой работающей версии продукта;
- уменьшение общего времени разработки;
- лучшие отношения с заказчиком;
- улучшение морального климата;
- увеличение шансов завершения проекта;
- более надежные оценки графика проекта;
- более аккуратные отчеты о состоянии;
- лучшее качество кода;
- меньшее количество документации.

Интеграция программ выполняется посредством *поэтапного* или *инкрементного* подхода.

**Поэтапная интеграция** состоит из этапов, перечисленных ниже:

1. «Модульная разработка»: проектирование, кодирование, тестирование и отладка каждого класса.
2. «Системная интеграция»: объединение классов в одну огромную систему.
3. «Системная дезинтеграция»: тестирование и отладка всей системы.

Проблема поэтапной интеграции в том, что, когда классы в системе впервые соединяются вместе, неизбежно возникают новые проблемы и их причины могут быть в чем угодно. Поскольку у вас масса классов, которые никогда раньше не работали вместе, виновником может быть плохо протестированный класс, ошибка в интерфейсе между двумя классами или ошибка, вызванная взаимодействием двух классов. Все классы находятся под подозрением.

Неопределенность местонахождения любой из проблем сочетается с тем фактом, что все эти проблемы вдруг проявляют себя одновременно. Это заставляет вас иметь дело не только с проблемами, вызванными взаимодействием классов, но и другими ошибками, которые трудно диагностировать, так как они взаимодействуют.

Поэтому поэтапную интеграцию называют еще «интеграцией большого взрыва»

Поэтапную интеграцию нельзя начинать до начала последних стадий проекта, когда будут разработаны и протестированы все классы. Когда классы, наконец, будут объединены и проявится большое число ошибок, программисты тут же ударятся в паническую отладку вместо методического определения и исправления ошибок.

Для небольших программ — нет, а для крошечных — поэтапная интеграция может быть наилучшим подходом. Если программа состоит из двух-трех классов, поэтапная интеграция может сэкономить ваше время, если вам повезет. Но в большинстве случаев инкрементный подход будет лучше.

При **инкрементной интеграции** вы пишете и тестируете маленькие участки программы, а затем комбинируете эти кусочки друг с другом по одному. При таком подходе — по одному элементу за раз — вы выполняете перечисленные далее действия:

1. Разрабатываете небольшую, функциональную часть системы. Это может быть наименьшая функциональная часть, самая сложная часть, основная часть или их комбинация. Тщательно тестируете и отлаживаете ее. Она послужит скелетом, на котором будут наращиваться мускулы, нервы и кожа, составляющие остальные части системы.
2. Проектируете, кодируете, тестируете и отлаживаете класс.
3. Прикрепляете новый класс к скелету. Тестируете и отлаживаете соединение скелета и нового класса. Убеждаетесь, что эта комбинация работает, прежде чем переходить к добавлению нового класса. Если дело сделано, повторяете процесс, начиная с п. 2.

Инкрементный подход имеет массу преимуществ перед традиционным поэтапным подходом независимо от того, какую инкрементную стратегию вы используете:

**Ошибки можно легко обнаружить** Когда во время инкрементной интеграции возникает новая проблема, то очевидно, что к этому причастен новый класс. Либо его интерфейс с остальной частью программы неправилен, либо его взаимодействие с ранее интегрированными классами приводит к ошибке. В любом случае вы точно знаете, где искать проблему.

**В таком проекте система раньше становится работоспособной** Когда код интегрирован и способен выполняться, даже если система еще не пригодна к использованию, это выглядит так, будто это скоро произойдет. При инкрементной интеграции программисты раньше видят результаты своей работы, поэтому их моральное состояние лучше, чем в том случае, когда они подозревают, что их проект может никогда не сделать первый вдох.

**Вы получаете улучшенный мониторинг состояния** При частой интеграции реализованная и нереализованная функциональность видна с первого взгляда. Менеджеры будут иметь лучшее представление о состоянии проекта, видя, что 50% системы уже работает, а не слыша, что кодирование «завершено на 99%».

**Вы улучшите отношения с заказчиком** Если частая интеграция влияет на моральное состояние разработчиков, то она также оказывает влияние и на моральное состояние заказчика. Клиенты любят видеть признаки прогресса, а инкрементная интеграция предоставляет им такую возможность достаточно часто.

**Системные модули тестируются гораздо полнее** Интеграция начинается на ранних стадиях проекта. Вы интегрируете каждый класс по мере его готовности, а не ожидая одного внушительного мероприятия по интеграции в конце разработки. Программист тестирует классы в обоих случаях, но в качестве элемента общей системы они используются гораздо чаще при инкрементной, чем при поэтапной интеграции.

**Вы можете создать систему за более короткое время** Если интеграция тщательно спланирована, вы можете проектировать одну часть системы в то время, когда другая часть уже кодируется. Это не уменьшает общее число человеко-часов, требуемых для полного проектирования и кодирования, но позволяет выполнять часть работ параллельно, что является преимуществом в тех случаях, когда время имеет критическое значение.

При поэтапной интеграции вам не нужно планировать порядок создания компонентов проекта. Все компоненты интегрируются одновременно, поэтому вы можете разрабатывать их в любом порядке — главное, чтобы они все были готовы к часу X.

При инкрементной интеграции вы должны планировать более аккуратно. Большинство систем требует интеграции некоторых компонентов перед интеграцией других. Так что планирование интеграции влияет на планирование конструирования — порядок, в котором конструируются компоненты, должен обеспечивать порядок, в котором они будут интегрироваться.

### **Нисходящая интеграция**

При нисходящей интеграции класс на вершине иерархии пишется и интегрируется первым. Вершина иерархии — это главное окно, управляющий цикл приложения, объект, содержащий метод main() в программе на Java, функция WinMain() в программировании для Microsoft Windows или аналогичные. Для работы этого верхнего класса пишутся заглушки. Затем, по мере интеграции классов сверху вниз, классы заглушек заменяются реальными.



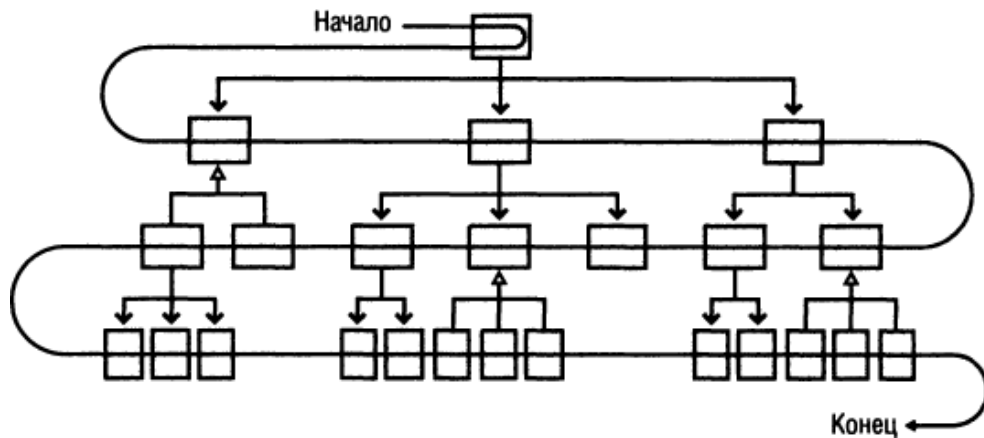
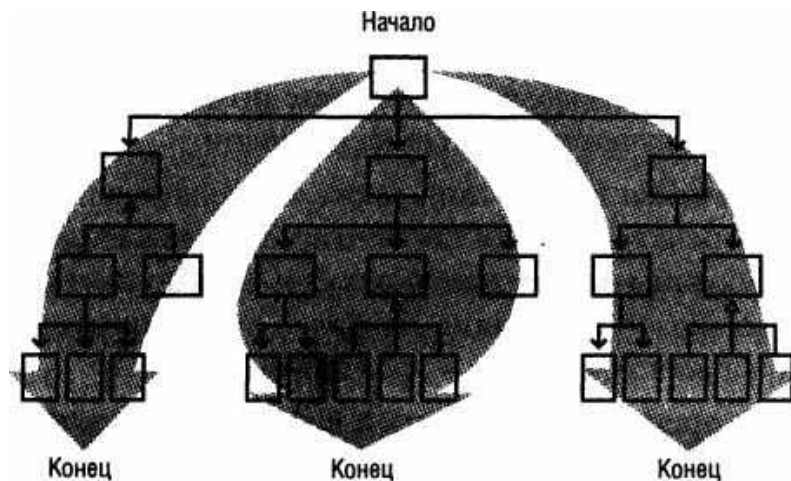


Рис.3 Нисходящая интеграция

При нисходящей интеграции вы создаете те классы, которые находятся на вершине иерархии, первыми, а те, что внизу, — последними.

Хорошей альтернативой нисходящей интеграции в чистом виде может стать подход с вертикальным секционированием.

Рис. 4 Вертикальное секционирование



При этом систему реализуют сверху вниз по частям, возможно, по очереди выделяя функциональные области и переходя от одной к другой.

#### **Восходящая интеграция**

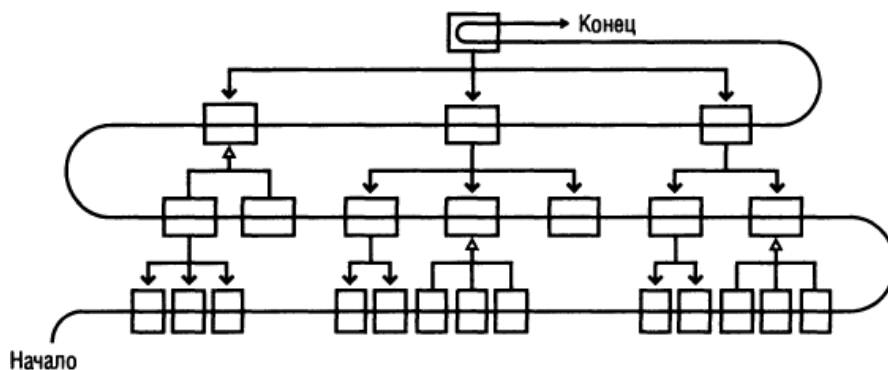


Рис.5 Восходящая интеграция

При восходящей интеграции вы пишете и интегрируете сначала классы, находящиеся в низу иерархии. Добавление низкоуровневых классов по одному, а не всех одновременно — вот что делает восходящую интеграцию инкрементной стратегией. Сначала вы пишете тестовые драйверы для выполнения низкоуровневых классов, а затем добавляете эти классы к тестовым драйверам, пристраивая их по мере готовности. Добавляя класс более высокого уровня, вы заменяете классы драйверов реальными.

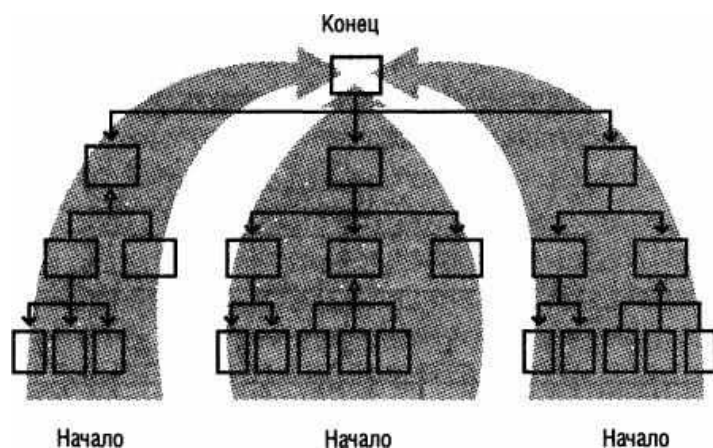


Рис. 6 Гибридный подход при восходящей интеграции

Как и нисходящую, восходящую интеграцию в чистом виде используют редко — вместо нее можно применять гибридный подход, реализующий секционную интеграцию.

### **Сэндвич-интеграция**

Проблемы с нисходящей и восходящей интеграциями в чистом виде привели к тому, что некоторые эксперты стали рекомендовать сэндвич-подход.

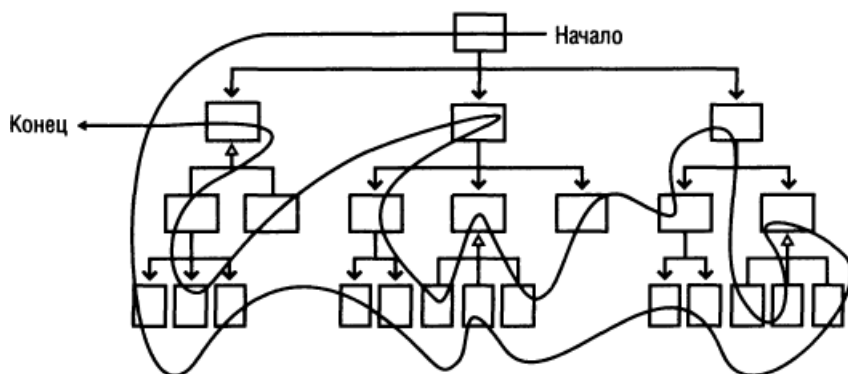


Рис. 7 Сэндвич-интеграция

Сначала вы объединяете высокоуровневые классы бизнес-объектов на вершине иерархии. Затем добавляете классы, взаимодействующие с аппаратной частью, и широко используемые вспомогательные классы в низу иерархии.

Напоследок вы оставляете классы среднего уровня.

### **Риск-ориентированная интеграция**

Риск-ориентированную интеграцию, которую также называют «интеграцией, начиная с самых сложных частей» (hard part first integration), похожа на сэндвич-интеграцию тем, что пытается избежать проблем, присущих нисходящей или восходящей интеграциям в чистом виде. Кроме того, в ней также есть тенденция к объединению классов верхнего и нижнего уровней в первую очередь, оставляя классы среднего уровня напоследок. Однако суть в другом.

При риск-ориентированной интеграции вы определяете степень риска, связанную с каждым классом. Вы решаете, какие части системы будут самыми трудными, и реализуете их первыми.

### **Функционально-ориентированная интеграция**

Еще один подход — интеграция одной функции в каждый момент времени. Под «функцией» понимается не нечто расплывчатое, а какое-нибудь поддающееся определению свойство системы, в которой выполняется интеграция.

Когда интегрируемая функция превышает по размерам отдельный класс, то «единица приращения» инкрементной интеграции становится больше отдельного класса. Это немного снижает преимущество инкрементного подхода в том плане, что уменьшает вашу уверенность об источнике новых ошибок. Однако если вы тщательно тестировали классы, реализующие эту функцию, перед интеграцией, то это лишь небольшой недостаток. Вы можете использовать стратегии инкрементной интеграции рекурсивно, сформировав сначала из небольших кусков отдельные свойства, а затем инкрементно объединив их в систему.

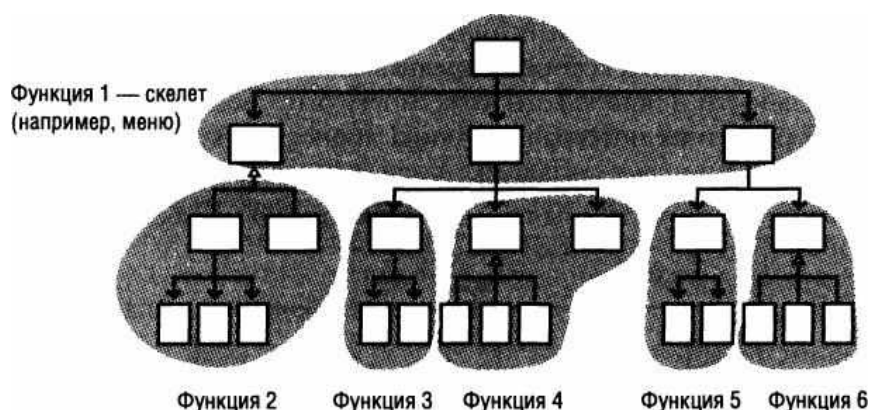


Рис. 8 Функционально-ориентированная интеграция

Обычно процесс начинается с формирования скелета, поскольку он способен поддерживать остальную функциональность. В интерактивной системе такой изначальной опцией может стать система интерактивного меню. Вы можете прикреплять остальную функциональность к той опции, которую интегрировали первой.

### Т-образная интеграция

Последний подход, который часто упоминается в связи с проблемами нисходящей и восходящей методик, называется «Т-образной интеграцией». При таком подходе выбирается некоторый вертикальный слой, который разрабатывается и интегрируется раньше других. Этот слой должен проходить сквозь всю систему от начала до конца и позволять выявлять основные проблемы в допущениях, сделанных при проектировании системы. Реализовав этот вертикальный участок (и устранив все связанные с этим проблемы), можно разрабатывать основную канву системы (например, системное меню для настольного приложения). Этот подход часто комбинируют с риск-ориентированной и функционально-ориентированной интеграциями.

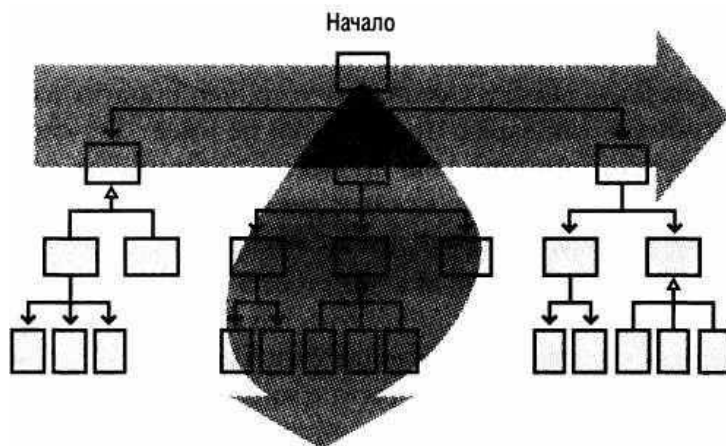


Рис. 9 Т-образная интеграция

### **Задание.**

1. Оформить внешнюю спецификацию.
2. Составить в виде блок-схемы алгоритм решения задачи.
3. Спроектировать и разработать модули программы для решения задачи на любом алгоритмическом языке программирования.
4. Выполнить отладку и тестирование модулей программы.
5. Выполнить инкрементную интеграцию модулей с использованием одного из подходов.
6. Выполнить системное тестирование программы.
7. 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 модулей программы.

5. Описание процесса интеграции модулей.

**Задача.** Задан двумерный массив размерности  $n \times m$ . Отсортировать элементы строк массива по возрастанию значений, а затем отсортировать строки массива по возрастанию среднего арифметического элементов строк.

Реализовать сортировку разными способами и сравнить эффективность этих способов для разных исходных данных.

**Контрольные вопросы.**

1. Значение фазы интеграции программных модулей.
2. Подходы к интегрированию программных модулей.  
Эффективность и оптимизация программ.

## **Лабораторная работа №21 «Тестирование интерфейса пользователя средствами инструментальной среды разработки»**

**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 (заголовок, 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.

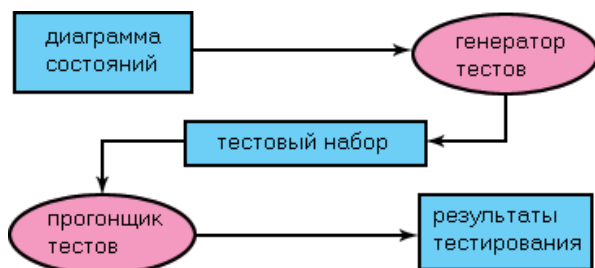


Рис. 2 Общая схема генерации и прогона тестов

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 №22 «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 №23 «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 эквивалентности. *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 №24 «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 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 — непереносимое 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 чем суть метода эквивалентных разбиений?
10. В чем суть метода анализа граничных значений?
11. В чем суть метода анализа причинно-следственных связей?

## **Лабораторная работа №25 «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.