Тестирование и отладка программного обеспечения

Лекция 6

Тема 6: Тестирование и отладка ПО

Перечень рекомендуемой литературы

- ***Смольянинов В.Ю.*** **Технологии и методы программирования. Ч.1: Технология программирования.** Курс лекций. М.: в/ч 33965, 2011-195 с.
- ***ГОСТ Р ИСО/МЭК 12119-2000.*** **Пакеты программ. Требования к качеству и тестирование.**
- ***Котляров В.П.*** **Основы тестирования программного обеспечения: Учебное пособие –** М. Бином, 2006 – 285 с.

Проверка ПО

- **Надежность ПО** – комплексное свойство ПО сохранять во времени в установленных пределах значения всех параметров, характеризующих способность ПО выполнять свои функции в заданных режимах и условиях эксплуатации
- Проверка правильности функционирования и соответствия предъявляемым требованиям
- Стадия разработки ПО непосредственно следующая за кодированием
- Цель проверки – выявление несоответствия программного обеспечения предъявляемым требованиям

Ошибка

- **Дефект (ошибка)** – несоответствие ПО предъявляемым требованиям
- **Типы ошибок:**
  - **Ошибки этапа компиляции** (синтаксические ошибки)
  - **Ошибки этапа выполнения** (семантические ошибки)
    - Ошибка (error) – неправильный результат
    - Отказ (failure) – невозможность работы
  - **Логические ошибки**
  - **Плавающая ошибка**

Ошибки этапа выполнения

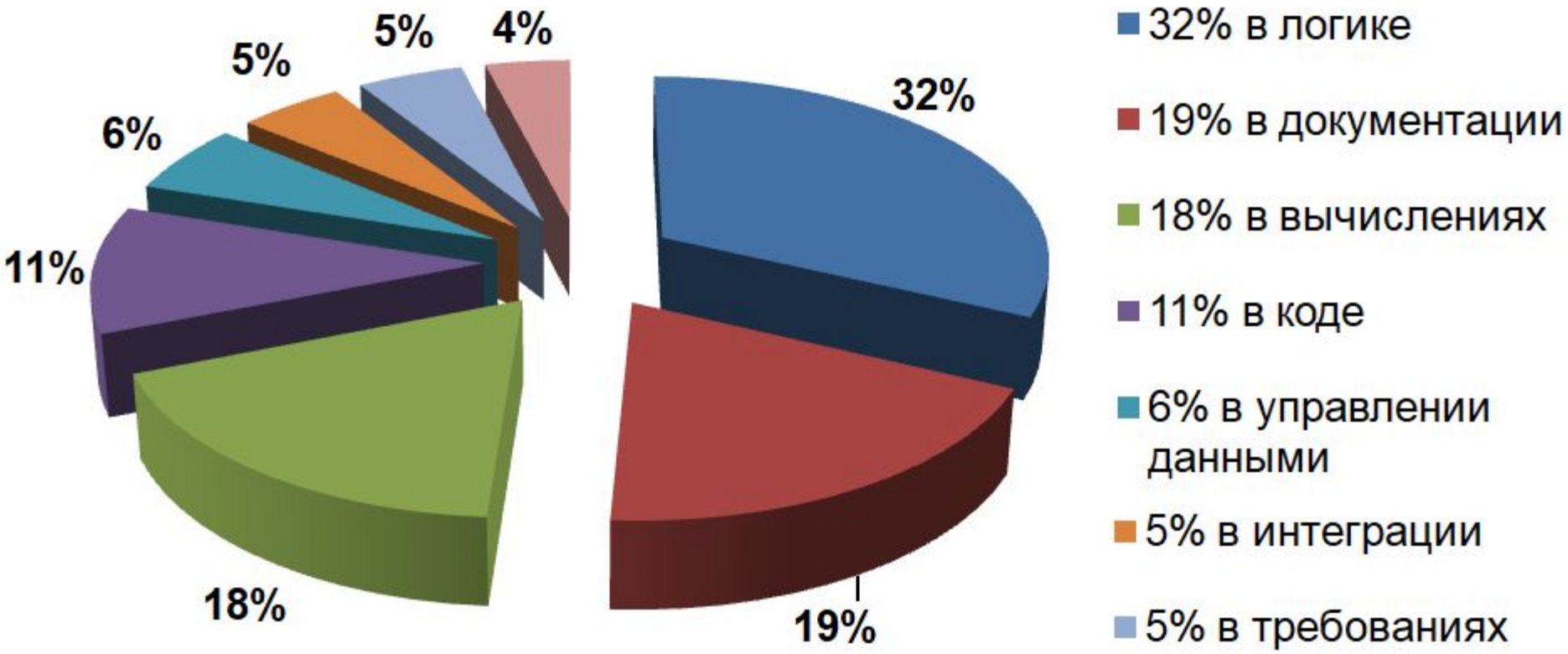
По международному стандарту ANSI/IEEE-729-83:

- Ошибка (error) – состояние программы, при котором выдаются неправильные результаты, причиной которых являются изъяны в операторах программы или в технологическом процессе ее разработки, приводящие к неправильной интерпретации входной информации
- Отказ (failure) – отклонение программы от функционирования или невозможность выполнять функции, определенные внешними спецификациями, что рассматривается как неработоспособное состояние программы.

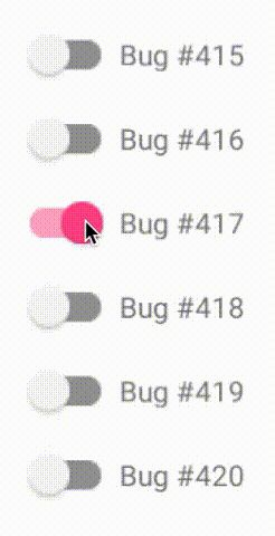
Плохой программист Джон сделал ошибку в коде, из-за которой каждый пользователь программы был вынужден потратить в среднем 15 минут времени на поиск обхода возникшей проблемы. Пользователей было 10 миллионов. Всего впустую потрачено 150 миллионов минут, то есть 2,5 миллиона часов. Если рабочий день человека 8 часов, то Джон уничтожил 312,5 тысяч человеко-дней, то есть около 856 человеко-лет. И если разделить это на среднюю продолжительность жизни 75 лет, значит получится, что Джон убил чуть больше 11 человек.

Как тебе спится, Джон – серийный программист?

Классификация ошибок



Примеры ошибок



- отсутствие инициализации переменных
- ошибки порядка вычислений
- проблемы с указателями
- неверная интерпретация сообщений компилятора о синтаксических ошибках
- «потеря» данных при неправильном значении указателя или индекса массива
- выход за границы массива
- неправильное использование аргументов функций
- переполнения стека или очереди

Проверка ПО

- **Проверка вручную** – без выполнения модулей и запуска ПО – анализ исходных текстов (*понимание программы*), возможно с использованием специальных средств проверки (*статический анализ кода*).
- **Проверка тестированием** – с выполнением модулей (*динамический анализ кода*).

Проверка вручную

- Проводится разработчиком или тестировщиком, **знающим алгоритм** работы программы, **логику** ее функционирования, **ориентирующимся** в исходных текстах
- Проводится путем **просмотра исходных текстов** с возможным использованием специальных средств проверки, упрощающих процесс выявления ошибок

Проверка тестированием

- Проводится тестировщиком, **не знающим логики** работы программы, а обладающим знаниями только о **внешних спецификациях**
- Выполняется весь набор тестов, чтобы проверить программное обеспечение на входных данных, составленных на основе внешних спецификаций.
- При выявлении ошибки – передача информации разработчику о выявленной ошибке для конкретных входных данных
- Разработчик проводит проверку вручную, но не полностью, а ориентируясь по логике работы программы, пытается определить место выявленной тестировщиком ошибки

Вручную

- Знать алгоритм
- Меньше ресурсов
- На этапе кодирования
- От 30 до 70% ошибок

Тестирование

- Знать внешнюю спецификацию
- Больше ресурсов (человек и времени)
- Готовые модули
- Ошибки выявляемые только тестированием

Виды проверок вручную

- Лексический и синтаксический контроль с использованием компиляторов (статический анализ кода)
- Инспекция модуля – изучения исходного текста модуля с целью выявления распространенных ошибок (требуется опыт программирования)
- Сквозной просмотр – обнаружение несоответствия между спецификацией программы и логикой работы
- Доказательство корректности – формальные средства соответствия логики работы программы и ее внешней спецификации

Ошибки, выявляемые вручную

- Неопределенное поведение программы – неинициализированные переменные, обращение к NULL-указателям
- Нарушение алгоритма использования функций и библиотек (отсутствие закрывающих операторов)
- Известные типовые сценарии небезопасного использования отдельных функций
- Переполнение буфера, выход за границы массива
- Ошибки кроссплатформенности (32 и 64)
- Ошибки повторяющегося кода
- Ошибки типов параметров функций

Тестирование

- **Тестирование –** процесс исследования программного обеспечения или его отдельных модулей путем выполнения с целью определения соответствия заданным требованиям (внешним спецификациям)

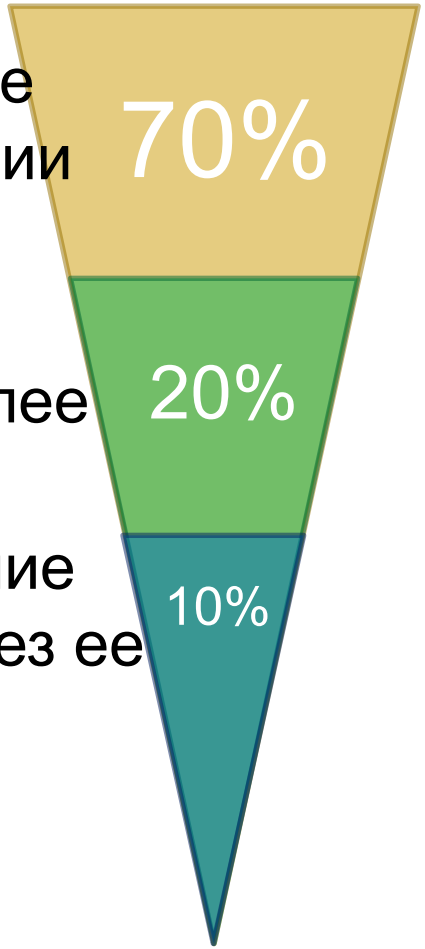
Требования + Проектирование	**40%**
Кодирование + Интеграция	**20%**
Тестирование	**40%**

Принципы тестирования

- Тестировщики – минимально заняты в разработке
- Разработчики тестов – опытные разработчики
- Разделение разработчиков тестов и тестировщиков
- Вероятность ошибки в части ПО тем выше, чем больше обнаружено уже в нем ошибок
- Тестирование должно всегда повторяться после внесения изменений
- Системность тестирования
- Тщательное проектирование тестов как для допустимых, так и для недопустимых ситуаций
- Целесообразна проверка побочных эффектов программы

Виды тестирования

- **По степени изолированности**
  - **Модульное тестирование** – автономное тестирование модуля, класса или функции в отрыве от всей системы
  - **Интеграционное тестирование** – одновременное тестирование двух и более модулей на совместимость
  - **Системное тестирование** – тестирование всей системы в целом, как правило, через ее внешний интерфейс;



Виды тестирования

- **По времени проведения**
  - **Тестирование новой функции** – вновь разработанный код (модульное тестирование)
  - **Приемочное тестирование** – тестирование, проводимое при приемке системы заказчиком
  - **Регрессионное тестирование** – последовательное тестирование системы в процессе ее разработки. Повторные итерации тестирования проводятся после исправления и модификации системы с целью проверки того, что изменения не ухудшили существующей функциональности и не внесли новые ошибки

Регрессионное тестирование

- Предназначено для проверки того, что исправления и модификации системы не ухудшили существующей функциональности и не внесли новые ошибки
- Тестирование проводится периодически в процессе разработки системы
- Варианты:
  - повторный запуск полного набора тестов
  - разработка регрессионного набора тестов, который получается из полного набора тестов путем исключения тестов, не затрагивающих измененные области кода и функциональные возможности

Регрессионное тестирование

Полный набор тестов	Регрессионный набор тестов
Прост в реализации	Требует дополнительные расходы на разработку
Дорогостоящий и неэффективный	Уменьшает расходы за счет исключения лишних тестов
Обнаруживает все ошибки	Может приводить к пропуску ошибок

- Стратегии построения регрессионных тестов:
  - активная – минимизация количества тестов с пренебрежением риска пропуска дефектов

  Снижение объема и времени тестирования, но и снижение вероятности обнаружения ошибок
  - консервативная – сохранение в наборе тестов, которые с ненулевой вероятностью обнаруживают дефекты

  Высокая вероятность обнаружения ошибок, но сохраняется большой объем тестов, соответственно требуются ресурсы и время на их проведение

Виды тестирования

- **По объекту тестирования:**
  - **Функциональное тестирование** - проверка того, что программа в целом ведет себя в соответствии с ожиданиями пользователя, формализованными в виде внешних спецификаций

Проверяются все функции системы с точки зрения ее пользователей

Критерии тестирования:

- покрытие тестами всех функциональных требований – система должна побывать во всех своих внутренних состояниях, пройдя по всем возможным переходам между состояниями
- покрытие тестами всех входных и выходных данных
- все классы входных данных должны корректно обрабатываться – на допустимых значениях должны быть получены достоверные результаты, недопустимые значения должны быть отброшены
- система должна сохранять стабильность при обработке недопустимых входных данных

Виды тестирования

- **По объекту тестирования:**
  - **Тестирование конфигурации –** определение возможности функционирования программной системы на различных конфигурациях оборудования

Тестируется:

- корректная работа с заявленным аппаратным обеспечением
- корректная работа с другими программными системами
- корректное поведение на изменение аппаратного или программного обеспечения
- корректная обработка проблем, возникающих в аппаратном или программном обеспечении

Виды тестирования

- **По объекту тестирования:**
  - **Тестирование безопасности –** проверка защиты информации, обрабатываемой программой

Тестируется:

- сохранность информации (защита от удаления, изменения, повреждения)
- возможность несанкционированного доступа

Виды тестирования

- **По объекту тестирования:**
  - **Тестирование удобства использования** – проверка удобства пользовательского интерфейса

Тестируются:

- время или количество необходимых действий для достижения определенной цели
- время или количество необходимых действий для доступа к нужной информации
- интерпретация ответов системы

Виды тестирования

- **По объекту тестирования:**
  - **Тестирование производительности –** определение того, что система обеспечивает должный уровень производительности при обработке пользовательских запросов

Выполняется при различных уровнях нагрузки на систему, на различных конфигурациях оборудования

Факторы, влияющие на производительность:

- количество поддерживаемых системой потоков
- количество свободных/занятых системных ресурсов
- количество свободных/занятых аппаратных ресурсов
- скоростные характеристики системы

Тестирование производительности

- Требования по производительности системы должны быть четко определены
- Требуются **генератор запросов** подающий на вход системы поток данных, типичных для сеанса работы с ней и **тестовое окружение** из программной и аппаратной компонент с возможностью моделирования различного уровня доступных ресурсов

Тестирование производительности

- **Нагрузочное тестирование** – тестирование системы на корректную работу с большими объемами данных, проверяются и определяются действующие пределы работоспособности системы
- **Стрессовое тестирование** – тестирование поведения системы при длительной непрерывной эксплуатации в условиях высокой нагрузки на систему, в том числе стрессовой нагрузки

Тестирование производительности

- Нагрузочное и стрессовое тестирование
- Задача – оценка производительности и устойчивости системы в случаях:
  - максимального выделения ресурсов, или их нехватки
  - подачи интенсивного входного потока продолжительный период времени
- Цель нагрузочного Т. – вывести систему из строя, определить пороговые условия, после которых она перестает функционировать.
- Цель стрессового Т. – определить время которое выдерживает система при стрессовой нагрузке
- Позволяют выявить узкие места в системе и направления ее дальнейших оптимизаций

Виды тестирования

- **По объекту тестирования:**
  - **Тестирование надежности и восстановления после сбоев –** проверка возможности системы восстанавливать свою функциональность и корректную работу после сбоев

Имитируются сбои аппаратного или программного обеспечения

Проверяется:

- минимизация потерь данных
- минимизация времени восстановления

Виды тестирования

- **По степени автоматизации:**
  - **Ручное тестирование** – запуск тестов и анализ результатов тестирования вручную
  - **Автоматическое** – запуск тестов и анализ результатов тестирования производится специальным программным обеспечением
  - **Автоматизированное** – запуск тестов автоматический, анализ результатов и принятие решения о дальнейшем проведении тестирования осуществляет человек

Стратегии проектирования тестов

По виду тестирования – по знанию внутреннего устройства программы

- **Черный ящик** – тестирование в условиях отсутствия информации о внутренней структуре программы, известна только внешняя спецификация

Тестирование с управлением по входным данным, то есть получение всех возможных выходных данных и результатов для всех возможных входных данных и воздействий

- **Белый ящик** – тестирование в условиях известного программного кода и логики работы программы
- Тестирование с управлением логикой программы на основе изучения текста программы, покрытия всех возможных состояний программы

Методы составления тестов

- По стратегии черного ящика
  - эквивалентных разбиений
  - граничных значений
  - функциональных диаграмм
  - предположения об ошибках

Метод эквивалентных разбиений

- Построение классов эквивалентности для входных и выходных данных
- Два теста (набора входных данных) являются эквивалентными, если совпадает множество ошибок, выявленных с их помощью

1) Разбиение всех возможных входных данных на конечное число правильных и неправильных классов эквивалентности

Метод эквивалентных разбиений

2) Выбираются тесты покрывающие максимальное количество правильных входных классов эквивалентности и правильных выходных классов, которые соответствуют правильным входным данным
3) Каждый следующий тест должен покрывать еще не покрытые классы эквивалентности
4) Для неправильных классов эквивалентности выбираются тесты по одному для каждого неправильного класса

- Формальной процедуры построения классов эквивалентности не существует!
- Существует опасность пропуска ошибок
- При выявлении ошибок в процессе их локализации необходимо перестраивать классы эквивалентности

Метод граничных значений

- Тесты покрывают не только все классы эквивалентности, но и проверяют программы для граничных значений классов эквивалентности.
- Выбор граничных значений правильных и неправильных классов эквивалентности

Метод граничных значений

- Выбор граничных значений правильных и неправильных классов данных:
  - если класс состоит из одного значения, то оно = граничному
  - если класс объединяет смежные значения, то граничное значение – лежащее на границе и ближайшее к границе
  - выбор в качестве граничного ближайшего значения, выходящего за границу класса
  - обязательная проверка максимальной и минимальной границы диапазона для входных и выходных данных
  - если классы эквивалентности строятся не только для значений, но и для количества элементов, то должны быть граничные значения и для количества элементов
  - если на входе/выходе присутствуют упорядоченная последовательность данных, то следует выбирать граничные значения для первых и последних элементов

Метод функциональных диаграмм

- Функциональная модель SADT на основе спецификации.
- Построение таблицы и ее минимизация (выбор тестов)

	Ф1	Ф2	Ф3	Ф4	Ф5
Класс эквивалентности 1	1				
Класс эквивалентности 2		1	1		
Класс эквивалентности 3	1			1	
Класс эквивалентности 4	1			1	
Класс эквивалентности 5					1

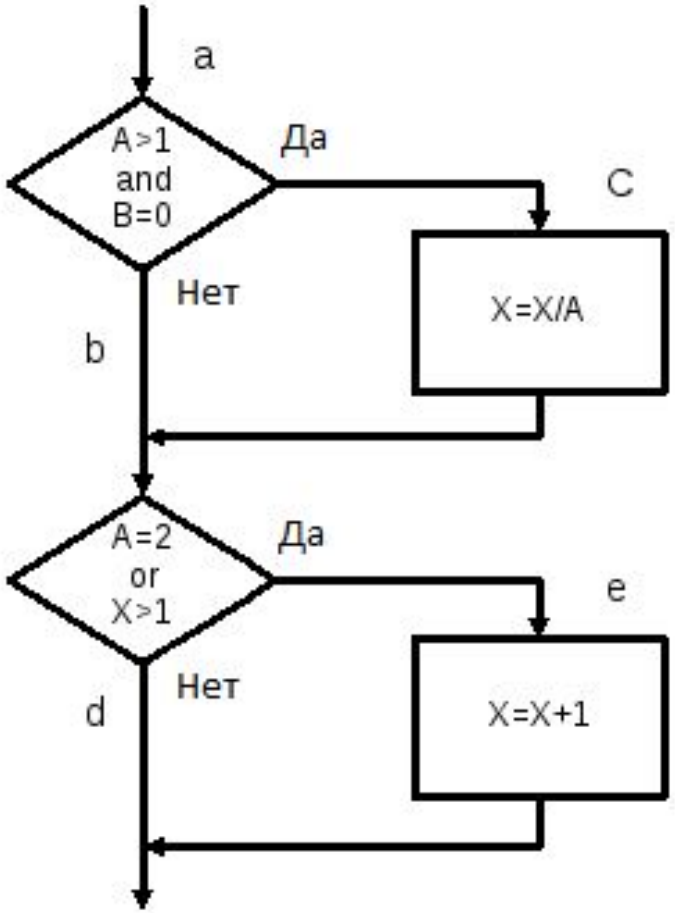
Метод предположения об ошибках

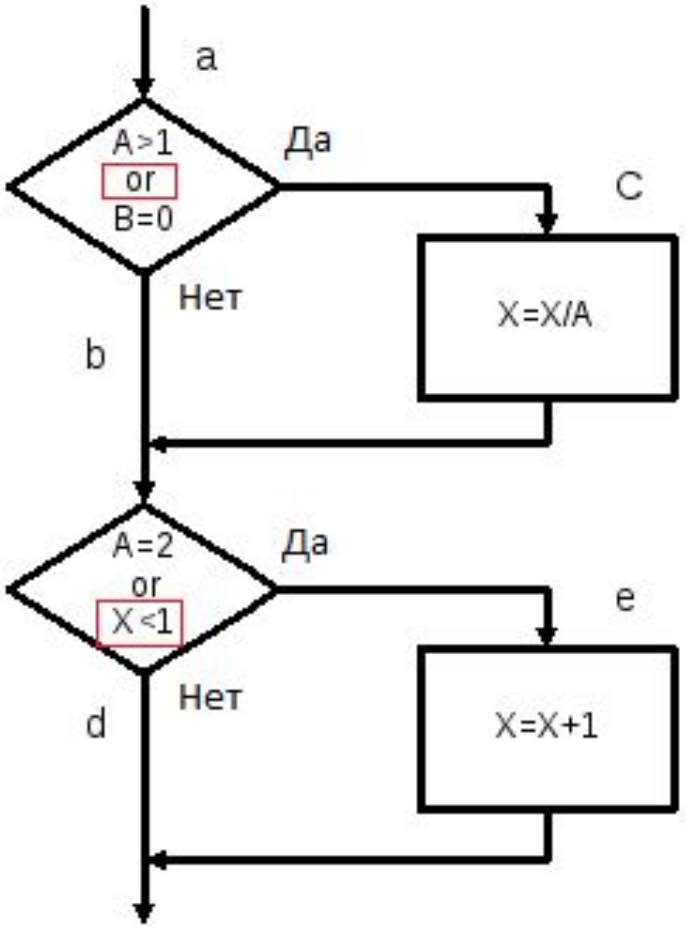
- Используется опыт специалистов по тестированию для предположения о возможных ошибках и проектирования соответствующих тестов

Методы составления тестов

- По стратегии белого ящика
  - покрытие операторов
  - покрытие решений
  - покрытие условий
  - покрытие решений и условий (покрытие логики)
  - комбинаторное покрытие условий

Стратегия белого ящика





Стратегия белого ящика

1. Покрытие операторов

- Набор тестов должен обеспечивать проверку выполнения каждого оператора хотя бы один раз

Тест	Путь	Ожидаемый результат	Фактический результат	Ошибка
A=2, B=0, X=3	*ace*	X=2,5	X=2,5	-

Стратегия белого ящика

2. Покрытие решений

- Набор тестов, при котором каждое решение примет как истинное, так и ложное значение
- Пример решения: (**A>1 and B=0)**

Тест	Путь	Ожидаемый результат	Фактический результат	Ошибка
A=3, B=0, X=3	*acd*	X=1	X=1	-
A=2, B=1, X=1	*abe*	X=2	X=1,5	+

Стратегия белого ящика

3. Покрытие условий

- Набор тестов, при котором все возможные результаты каждого условия в решении выполнялись хотя бы один раз
- Пример условия: **(A>1)**

Тест	Путь	Ожидаемый результат	Фактический результат	Ошибка
A=2, B=0, X=4	*ace*	X=3	X=3	-
A=1, B=1, X=0	*abd*	X=0	X=1	+

Стратегия белого ящика

4. Покрытие решений и условий (логики).

- Набор тестов, при котором результаты каждого решения выполняются хотя бы один раз и результаты каждого условия в решении выполняются хотя бы один раз

- не всегда можно проверить все условия, например, скрытые за другими условиями
- Низкая чувствительность к ошибкам в логических выражениях (первое решение содержит AND – хоть одно условие =0, тогда остальные не проверяются)

Стратегия белого ящика

5. Комбинаторное покрытие условий

- Набор тестов, при котором все возможные комбинации результатов условия в каждом решении выполнялись по крайней мере один раз
  - а) A>1, B=0
  - б) A>1, B≠0
  - в) A≤1, B=0
  - г) A≤1, B≠0

  - д) A=2, X>1

Стратегия белого ящика

5. Комбинаторное покрытие условий

Тест	Путь	Ожидаемый результат	Фактический результат	Ошибка
A=2, B=0, X=4	*ace*	X=3	X=3	-
A=2, B=1, X=1	*abe*	X=2	X=1,5	+
A=0,5 B=0, X=2	*abe*	X=3	X=4	+
A=0.5, B=1, X=1	*abd*	X=1	X=1	-

Критерии выбора тестов

- **Критерий должен быть достаточным** – показывать, когда некоторое множество тестов достаточно для тестирования программы
- **Критерий должен быть полным** – для любой ошибки должен существовать тест в наборе тестов, удовлетворяющих критерию, обнаруживающий ошибку
- **Критерий должен быть надежным** – любые два множества тестов, удовлетворяющих критерию, должны одновременно раскрывать или не раскрывать одни и те же ошибки
- **Критерий** должен быть легко проверяемым

Мутационные критерии

- ***Мутации*** – мелкие ошибки в программе
- ***Мутанты*** – программы, отличающиеся друг от друга *мутациями*
- **Метод мутационного тестирования** – в разрабатываемую программу P вносят *мутации*, т.е. искусственно создают программы-мутанты P1, P2... Затем программа P и ее *мутанты* тестируются на одном и том же наборе тестов T
- Если на наборе T подтверждается правильность программы P и, кроме того, выявляются все внесенные в программы-мутанты ошибки, то **набор тестов T соответствует мутационному критерию**, а тестируемая программа объявляется **правильной**
- Если некоторые *мутанты* не выявили всех *мутаций*, то надо расширять набор тестов T и продолжать тестирование. Так как T не удовлетворяет критерию.

Структурные критерии

- Используется информация о структуре программы - тестирование по стратегии «белого» ящика
- **Критерий тестирования операторов** – набор тестов в совокупности должен обеспечить прохождение каждого оператора не менее одного раза
- **Критерий тестирования ветвей** – набор тестов в совокупности должен обеспечить прохождение каждой ветви программы не менее одного раза
- **Критерий тестирования путей** – набор тестов в совокупности должен обеспечить прохождение каждого пути выполнения программы не менее одного раза

Функциональные критерии

- Формулируются в описании требований к программе, тестирование по стратегии «черного» ящика
- Обеспечивают контроль степени выполнения требований заказчика к программе и взаимодействие ее с окружающей средой
- **Критерий тестирования пунктов спецификации** – набор тестов в совокупности должен обеспечить проверку каждого тестируемого пункта не менее одного раза
- **Критерий тестирования классов входных/выходных данных** – набор тестов в совокупности должен обеспечить проверку представителя каждого класса входных/выходных данных не менее одного раза

Функциональные критерии

- **Критерий тестирования правил** – если существует набор правил, описывающих входные и выходные данные, то набор тестов в совокупности должен обеспечить проверку каждого правила
- **Критерий тестирования функций** – набор тестов в совокупности должен обеспечить проверку каждой описанной функции программы или модуля не менее одного раза
- **Комбинированный критерий** – набор тестов в совокупности должен обеспечить проверку всех комбинаций непротиворечивых условий **программы и спецификации** не менее одного раза. При этом все комбинации непротиворечивых условий должны быть подтверждены, а условия противоречий должны быть обнаружены

Стохастические критерии

- Применяется при тестировании сложных программных комплексов, когда количество тестов является слишком большим или полный набор тестов невозможно разработать и исполнить на этапе тестирования программного обеспечения
- Порядок тестирования:

1. Разрабатывается программа-имитатор случайных последовательностей входных данных {X}
2. Независимым образом вычисляется множество выходных значений {Y} для соответствующих входных значений и формируется тестовый набор {(X,Y)}
3. При невозможности вычисления выходных значений должно быть определено некоторое правило или распределение выходных значений F(X)

Стохастические критерии

4. Проводится тестирование приложения с использованием программы–имитатора, при этом проверяется либо принадлежность результата известному множеству выходных значений {Y}, либо соответствие правилу или распределению F(X).
5. Окончание тестирования:
   - Статистические методы – решение о прекращении тестирования принимается на основании совпадения гипотез о распределении случайных величин
   - Методы оценки скорости выявления ошибок – тестирование прекращается, если интервал времени между текущей ошибкой и следующей слишком велик для фазы тестирования приложения или больше определенного порога
   - Достижение пороговых ограничений: количество затраченных человеко-часов или календарного времени, выделенного на этап тестирования

Модульное тестирование

Модульное тестирование – тестирование отдельных модулей, классов, функций программы в отрыве от всей программы в целом

- Стратегия белого ящика
- Исходные данные – внешняя спецификация и исходный код
- Применяется в процессе разработки модулей, а также после завершения разработки перед сборкой системы и тестирования ее в целом

Модульное тестирование

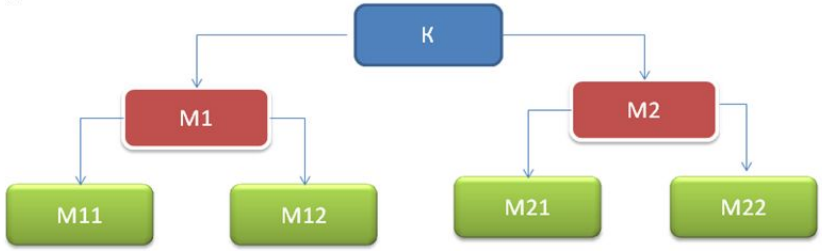
- ошибку проще найти в модуле, чем в системе в целом
- возможность параллельного тестирования модулей в крупной системе
- хорошо обнаруживаются внутренние ошибки модуля
- плохо обнаруживаются ошибки интерфейсов модуля
- необходимость разработки окружения тестирования модуля – дополнительных драйверов и заглушек для всех интерфейсов модуля
- опасность внесения дополнительных ошибок в разработанное окружение тестирования, не позволяющее корректно протестировать модуль
- возможная сложность разработки окружения тестирования

Интеграционное тестирование

Интеграционное тестирование – одновременное тестирование двух и более модулей на совместимость

- Тестирование интерфейса взаимодействия
- Стратегия «белого ящика» на модульном уровне, основываясь на знании взаимосвязи тестируемых модулей, внутренняя структура модуля не учитывается
- Объект тестирования – не функции модуля, а сами модули
- Основное сосредоточение на выявлении ошибок в **интерфейсах** между тестируемыми модулями, использовании **глобальных переменных**
- Применяется на этапе сборки оттестированных модулей в единый комплекс

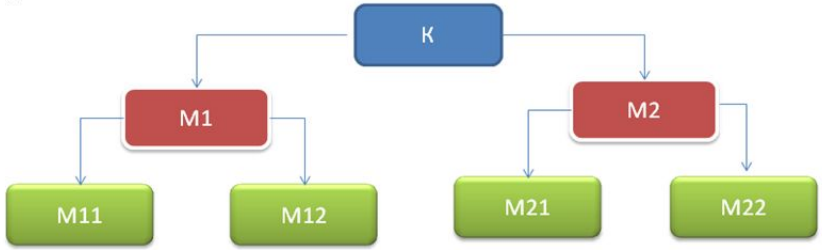
Интеграционное тестирование



Методы сборки модулей:

- **Монолитный** – одновременное объединение всех модулей
  - возможность распараллеливания работ на начальной фазе тестирования и тестирования каждого модуля по отдельности
  - большие трудозатраты, связанные с разработкой окружения тестирования модулей
  - сложность локализации ошибок в собранном комплексе
  - сложность тестирования межмодульных интерфейсов

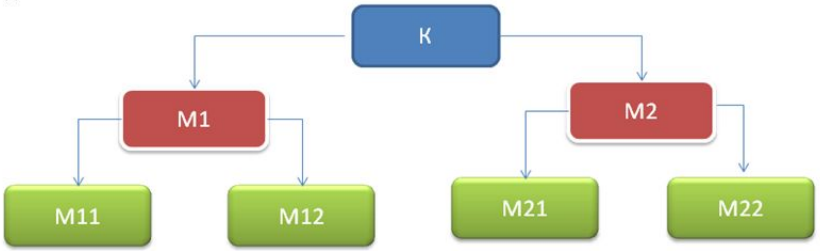
Интеграционное тестирование



Методы сборки модулей:

- **Инкрементальный** – пошаговая (помодульная) сборка комплекса с пошаговым тестированием комплекса
  - легкость локализации ошибок по сравнению с монолитным тестированием за счет постепенного наращивания объема тестируемого кода
  - тестирование межмодульных интерфейсов
  - сложность распараллеливания работ

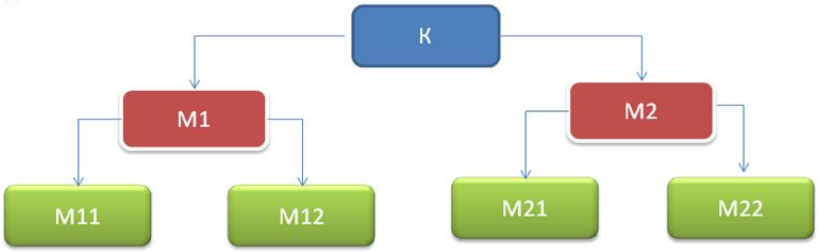
Интеграционное тестирование



Инкрементальный метод:

- **Сверху-вниз** (нисходящее тестирование) – тестирование начинается с основного модуля с использованием заглушек вместо невовлеченных в тестирование модулей
  - Сосредоточение на поведении тестируемого комплекса в целом
  - Раннее выявление концептуальных ошибок в алгоритме работы всего комплекса
  - Выделение приоритетных для тестирования модулей
  - Высокая потребность в заглушках, работающих в различных режимах тестирования комплекса
  - Параллельная разработка модулей верхних и нижних уровней приводит к не всегда эффективной реализации модулей из-за подстройки не тестированных модулей нижних уровней к уже оттестированным модулям верхних уровней

Интеграционное тестирование



Инкрементальный метод:

- **Снизу-вверх** (восходящее тестирование) – тестирование начинается с модулей низшего уровня, которые потом включаются в тестирование модулей более высокого уровня
  - Низкая потребность в заглушках
  - Запаздывание проверки концептуальных особенностей тестируемого комплекса в целом
- **Комбинированный способ** – одновременное нисходящее и восходящее тестирование
  - Позволяет сохранить плюсы и избавиться от минусов
  - Характерно для больших систем

Системное тестирование

Системное тестирование – тестирование системы в целом как единого объекта тестирования, проводящееся после интеграционного тестирования, которое устранило внутренние ошибки и ошибки межмодульных интерфейсов

- Внимание на поведении системы и ее внешних интерфейсах
- Стратегия «черного ящика»
- Отдельный коллектив тестировщиков, не участвовавших в процессе разработки

Этапы тестирования

1. Тестирование модулей
2. Тестирование сопряжений модулей (интеграционное)

3.1 Тестирование внешних функций (системное функциональное)

3.2 Системное тестирование (безопасность, производительность, надежность и т.п.)

4.1 Приемочное тестирование

4.2 Тестирование конфигурации.

Тесты

- **Тест** – задача, имеющая входные данные и соответствующие им известные выходные данные
- Тестирование проводится с использованием **набора тестов**, поскольку один тест не может выполнить проверку всей программы
- Тестирование проводится путем сравнения полученного результата с заранее известным для определенного набора входных данных

Тесты

Исходная информация – требования и алгоритм

Требования :

- Функциональные – явно описывают, что система должна делать и какие выполнять преобразования входных данных в выходные.
- Нефункциональные – определяют свойства системы, напрямую не связанные с функциональностью, например – время отклика на действие пользователя, время непрерывной работы …

Типы тестов

- **допустимые данные** – проверяют корректность работы программы, основных ее алгоритмов на соответствие спецификации на корректных данных
- **граничные данные** – проверяют поведение программы на входных данных, находящихся на границе классов эквивалентности, либо значения переменных являются предельными для соответствующего типа, максимальная длина строки и т.п.
- **отсутствие данных** – проверяют наличие дефектов, проявляющихся при отсутствии данных на входе программы
- **повторный ввод данных** – проверка одинакового результата при вводе одних и тех же данных

Типы тестов

- **неверные данные** – проверка поведения системы при подаче некорректных данных или данных неверного размера, символов вместо чисел, числа вне допустимого диапазона
- **реинициализация системы** – проверка поведения системы при ее повторной инициализации
- **устойчивость системы** – способность выдерживать нештатную нагрузку при большом входном потоке данных или продолжительной работе
- **нештатные состояния среды выполнения** – проверка поведения системы при изменении окружающей среды – исчерпывание памяти, нехватка процессорного времени. Система должна корректно завершить или приостановить свою работу

Управление тестированием

- Тестирование не только выявляет ошибки, но и позволяет предупреждать их появление в дальнейшем за счет:
  - Проектирования тестов
  - Тестирование(проверка) на всех этапах ЖЦ
  - Тестированию подлежат:
    - Программа
    - Требования к программе
    - Архитектура
    - Исходный код
    - ТЕСТЫ
- Это позволяет организовать управляемый процесс тестирования

Управление и документирование тестирования

- Системность тестирования
- Документирование – основа **управляемости** и **повторяемости** процесса тестирования
- Назначение документации - координация совместных действий большого количества разработчиков в течение более или менее длительных промежутков времени
- Документация тестирования ведется на всех этапах разработки ПО
- Виды документации:
  - План тестирования
  - Результаты тестирования

План тестирования

План тестирования – документ, определяющий стратегию тестирования, содержащий:

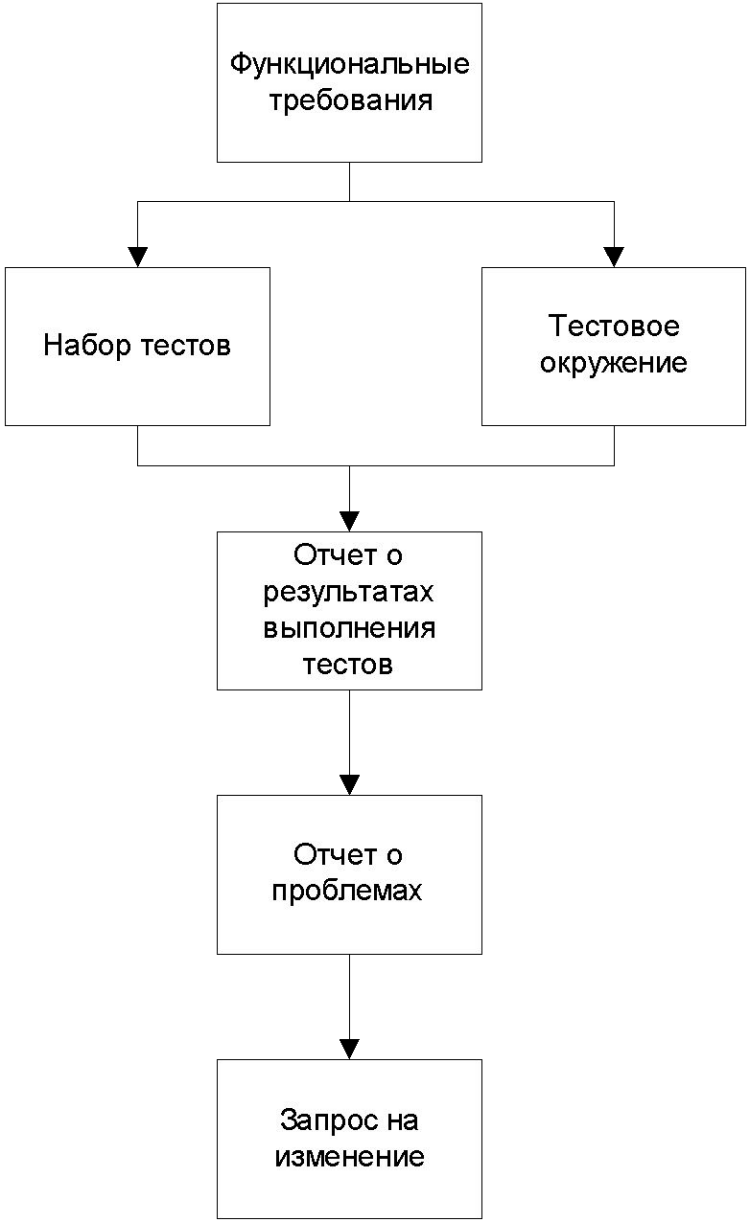
1. цели тестирования
2. объекты тестирования (функции, элементы и т.п.)
3. этапы и задачи тестирования
4. сроки и ответственность за выполнение задач и этапов
5. ресурсы, необходимые для проведения тестирования на каждом этапе
6. инструментальные средства, необходимые для проведения тестирования на каждом этапе
7. правила оформления тестов и их результатов
8. подходы к тестированию – (методики тестирования)
9. критерии успешности/неуспешности, начала/окончания тестирования
10. перечень тестов, их назначение и порядок выполнения

Результаты тестирования

- результаты выполнения тестов, позволяющие без их повторного проведения судить об успешности или неуспешности их выполнения
- выходные данные для соответствующих входных данных
- выявленные аномалии в поведении системы при тестировании

На основании результатов тестирования – выводы о:

- корректности работы системы
- соответствии системы внешним спецификациям
- наличии определенных дефектов, требующих устранения


Функциональные требования
Набор тестов
Тестовое окружение
Отчет о результатах выполнения тестов
Отчет о проблемах
Запрос на изменение

Отладка

- **Тестирование** = выявление дефектов
- **Отладка** = тестирование + локализация и устранение дефектов
- **Процесс отладки:**

1. Стабилизация ошибки – обеспечение гарантированного воспроизведения ошибки
2. Локализация ошибки – определение источника ошибки
3. Исправление ошибки
4. Тестирование исправления
5. Поиск схожих ошибок

Методы локализации ошибок

- **Аналитические –** основаны на результатах тестирования и анализе текста программы
- **Экспериментальные (инструментальные) –** основаны на использовании специальных средств отладки (отладчики)

Аналитический метод

1. Сбор всевозможной информации об ошибке, действиях и данных приводящих к возникновению ошибки
2. Анализ собранных данных и выявление закономерностей в появлении ошибки (*при необходимости проведение дополнительного тестирования и п.1.*)
3. Формирование гипотез об ошибке, выбор одной гипотезы путем повторного выполнения п.п.1-2.
4. Проверка гипотезы – подтверждение или опровержение гипотезы путем тестирования или инспектирования программы, при опровержении гипотезы – сбор дополнительных данных и формулирование новой гипотезы (п.п. 1-3)
5. Гипотеза подтвердилась – ошибка найдена

Экспериментальный метод

- средства аварийной печати – выдача информации при возникновении аварийной ситуации
  - выдача на экран текущего состояния регистров процессора, сохранения dump программы, позволяющие проводить последующий разбор произошедшего
  - встраивание разработчиком в программу средств печати вспомогательной информации (*логические состояния программы или объекта, значения переменных)*

Экспериментальный метод

- средства трассировки и слежения – отслеживание информации в процессе выполнения программы о состоянии вычислений с использованием отладчика
  - арифметическая трассировка – отслеживание значений переменных
  - логическая трассировка – отслеживание порядка выполнения программы (ветвей алгоритма, порядка вызова процедур и т.п.)
  - трассировка – пошаговое выполнение программы, когда программа выполняется построчно, если выполняемая строчка содержит вызов функции, то в зависимости от режима трассировки возможен заход в тело функции или выполнение функции целиком и остановка на следующей строке кода
  - слежение – выполнение программы до возникновения определенных условий или попадания в определенный этап вычислений и переход в режим трассировки (контрольные точки)

Экспериментальный метод

- средства динамического контроля – генерация компилятором такого программного кода, который осуществляет дополнительный контроль состояния вычислений в процессе выполнения программы, например средства реверсивного выполнения – сохранение на каждом шаге программы всех значений переменных. При возникновении ошибки можно в обратном порядке, двигаясь от следствия к причине локализовать ошибку

Экспериментальный метод

- средства печати в узлах выполнения программы – вставка в текст дополнительных операторов вывода информации о состоянии вычислений при нахождении программы на определенном шаге вычислений

Принципы экспериментального метода

- Планирование
- Минимальное количество данных при аварийной печати или выводе информации на экран
- Автоматизации процесса отладки
- Сохранение отладочных средств в тексте программы

Принципы исправления ошибок

- Обнаруженная ошибка либо исправляется немедленно, либо фиксируется факт наличия ошибки с принятием решения об откладывании действий по исправлению
- Исправление ошибки требует возврата к тому этапу разработки ПО, на котором она была допущена
- Вероятность правильного исправления ошибки ≠ 100%
- Чем крупнее программный продукт, тем больше вероятность внесения новых ошибок при исправлении существующих
- Документирование исправлений – изменения вносятся не только в программный код, но и в сопутствующую документацию (заголовки модулей, комментарии, техническую документацию)
- Исправление ошибок, а не их симптомов

Анализ результатов отладки

- При анализе результатов отладки необходимо ответить на следующие вопросы:
  - на каком этапе разработки ПО была совершена ошибка
  - кто автор ошибки
  - каковы причины появления ошибки
  - почему данная ошибка была обнаружена только сейчас
  - каким образом была обнаружена ошибка – используемые методы и средства тестирования
  - тип ошибки
  - проявления ошибки
- Это даст возможность:
  - повысить качество системы при последующем сопровождении
  - повысить квалификацию соответствующего коллектива разработчиков
  - повысить качество последующих разрабатываемых систем
  - принять решение об используемых инструментальных средствах
  - принять решение об организации процесса разработки ПО

Специфика отладки систем реального времени

- **Система реального времени** – система, которая должна реагировать на события во внешней по отношению к системе среде или воздействовать на среду в рамках требуемых временных ограничений **Время реакции** определяется при создании системы
- Корректность функционирования =

  =корректность вычислений + время результата
  - **системы жесткого реального времени** – не допускают задержек реакции системы ни при каких условиях (результаты бесполезны)
  - **системы мягкого реального времени** – задержка реакции не критична, но увеличивает стоимость результатов или снижению производительности системы

Отладка СРВ

- Более сложный процесс чем для диалоговых систем
  - Не ждут действий пользователя
  - Меняют свое состояние
- Основная сложность – недетерминированность поведения СРВ – при одних и тех же входных данных может быть разный результат

Методы отладки СРВ

- Активная отладка
  - отладчик имеет право останавливать выполнение системы и начинать или продолжать выполнение с некоторого адреса, отличного от точки останова, изменять значение переменных и регистров
  - возможность внесения сбоев в нормальную работу системы в связи с установленными временными ограничениями
  - возможность корректировки процесса выполнения

Методы отладки СРВ

- Мониторинг
  - сбор данных о работе системы (значения регистров, переменных, стадии работы системы, происходящие события) с использованием средств отладки, встроенных в процессе разработки ПО
  - полученные данные анализируются или используются при моделировании поведения системы на тестовой машине с использованием активного отладчика
  - основанная система заменяется на макет после тестирования и отладки макета на стенде
  - невмешательство в работу системы
  - невозможность изменения процесса выполнения

Тестирование и отладка программного обеспечения

Лекция 6

Тема 6: Тестирование и отладка ПО

Вопросы

1. Проверка ПО. Виды проверок. Ошибка определение и классификация.
2. Классификация и виды тестирования ПО.
3. Тестирование ПО по стратегии «черного ящика». Методы составления тестов.
4. Тестирование ПО по стратегии «белого ящика». Методы составления тестов.
5. Критерии выбора тестов.
6. Определение теста. Типы тестов. Управление и документирование процесса тестирования – план и результаты тестирования.
7. Тестирование и отладка ПО. Основные шаги процесса отладки. Методы локализации ошибок. Специфика отладки систем реального времени.