Тестирование и отладка программных средств

Лекция 10

Понятие тестирования

- *Тестирование* – это проверка соответствия свойств программного продукта спецификации требований
- Основным приемом тестирования является выполнение программ на некотором наборе данных, для которого заранее известен получаемый результат или известны правила поведения этих программ

Тестовые наборы

- Совокупность исходных данных и ожидаемого результата называется *тестовым вариантом* или просто *тестом*
- Каждый тест представляет собой вариант взаимодействия с тестируемой системой и проверки корректности ее поведения
- Хорошим считается тест, обеспечивающий высокую вероятность обнаружения ошибки

Тестовые наборы

- Каждый тест представляет собой вариант взаимодействия с тестируемой системой и проверки корректности ее поведения
- Хорошим считается тест, обеспечивающий высокую вероятность обнаружения ошибки

Успешность тестирования

- Что считать удачным исходом тестирования?
- С точки зрения тестировщика – это обнаружение какого-либо несоответствия требованиям (ошибки при выполнении функции, недостаточной производительности, низкого качества пользовательского интерфейса)
- С точки зрения разработчика, напротив, - отсутствие выявленных дефектов

Полное тестирование

- *Полным* или *исчерпывающим* тестированием называется проверка всех вариантов взаимодействия с системой
- Это идеальный случай, который, разумеется, не встречается на практике
- Утверждение о правильности программы, т.е. о ее полном соответствии спецификациям требований, можно сделать только по результатам исчерпывающего тестирования

Тестовое покрытие

- Практически оценивается только степень соответствия программы ее спецификации
- Таким образом, можно лишь утверждать, что такое соответствие имеет место с определенной вероятностью
- Для оценки степени полноты тестирования вводится понятие *уровня тестового покрытия*

Тестовое покрытие

- Иначе говоря, уровень тестового покрытия определяет степень охвата данным тестовым набором различных вариантов взаимодействия с программным средством

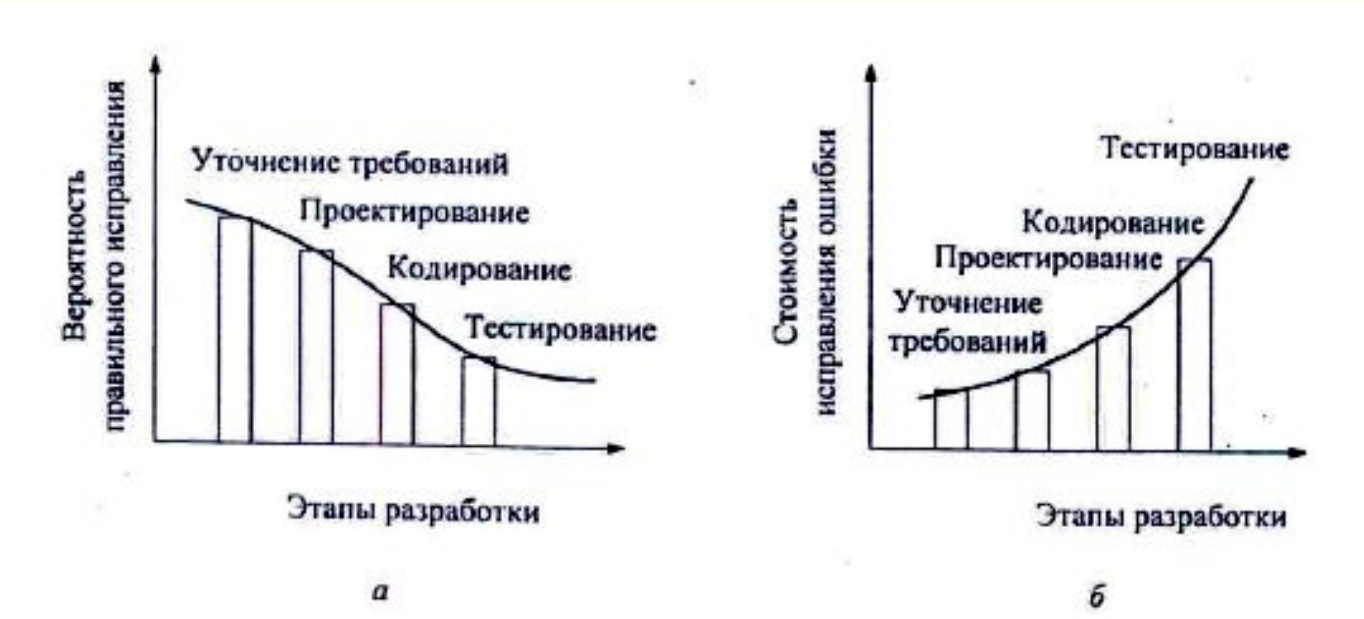
Понятие отладки

- *Отладка* − это деятельность, направленная на обнаружение причины возникновения того или иного дефекта программного продукта и на ее устранение
- Тестирование и отладка – это тесно связанные, но разные виды деятельности
- Далее речь, в основном будет идти о тестировании соответствия программы функциональным требованиям, т.е. о поиске ошибок в выполнении функций

Раннее тестирование

- Никакое тестирование не способно обнаружить всех ошибок в программе, но правильная организация этого процесса может существенно сократить их число
- Большинство моделей жизненного цикла предусматривает начало тестирования уже на ранних стадиях процесса разработки
- Это объясняется тем обстоятельством, что чем раньше обнаружена ошибка, тем легче и дешевле ее исправить

Раннее тестирование



Классические методы тестирования

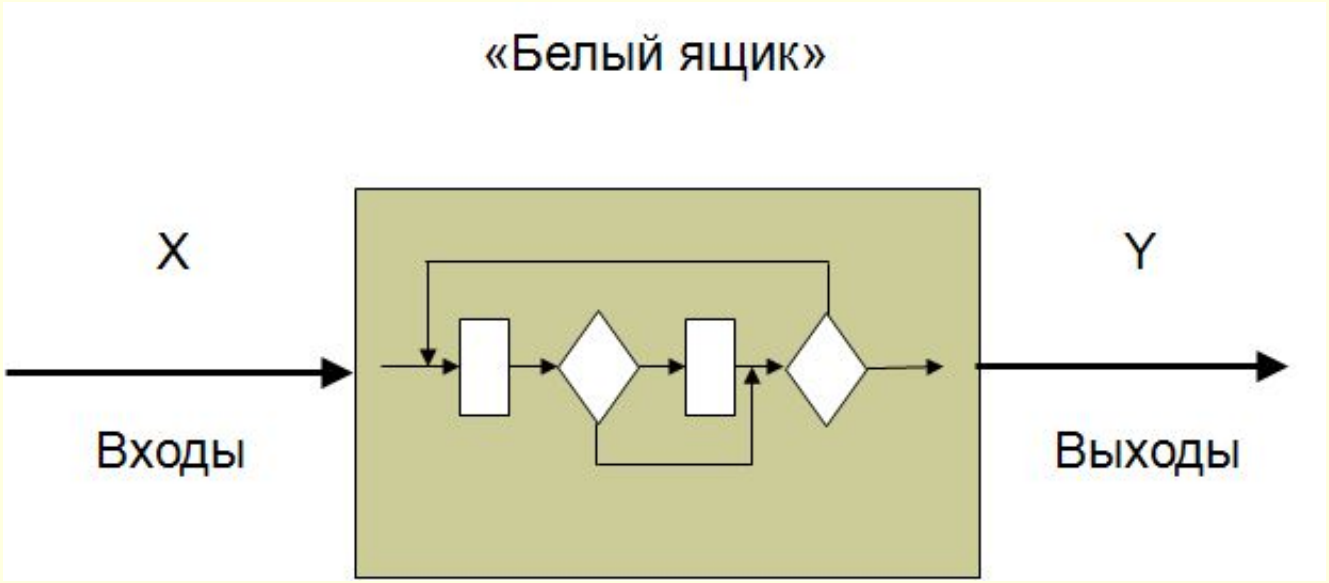
- Основополагающие принципы тестирования были разработаны в рамках структурного подхода к созданию программных средств
- Соответствующие им методы тестирования получили название *классических*

Формирование тестов

- Соответственно, существуют два принципиально различных подхода к формированию тестовых наборов:
  - *функциональный,*
  - *структурный*

СТРУКТУРНОЕ ТЕСТИРОВАНИЕ

Структурное тестирование



Базируется на знании внутренней логической структуры тестируемого ПС вплоть до уровня исходных текстов

Назначение

- Основное назначение структурного тестирования – проверка внутренней логики ПС
- Структурные тесты проверяют:
  - корректность построения отдельных элементов и правильность их взаимодействия
  - управляющие и информационные связи между элементами программы

Формирование тестов

- Тесты формируются на основе анализа внутренней структуры программы
- Одним из способов фиксации этой структуры является *потоковый граф:*
  - узлы графа соответствуют операторам или предикатам;
  - дуги графа отображают потоки управления в программе;

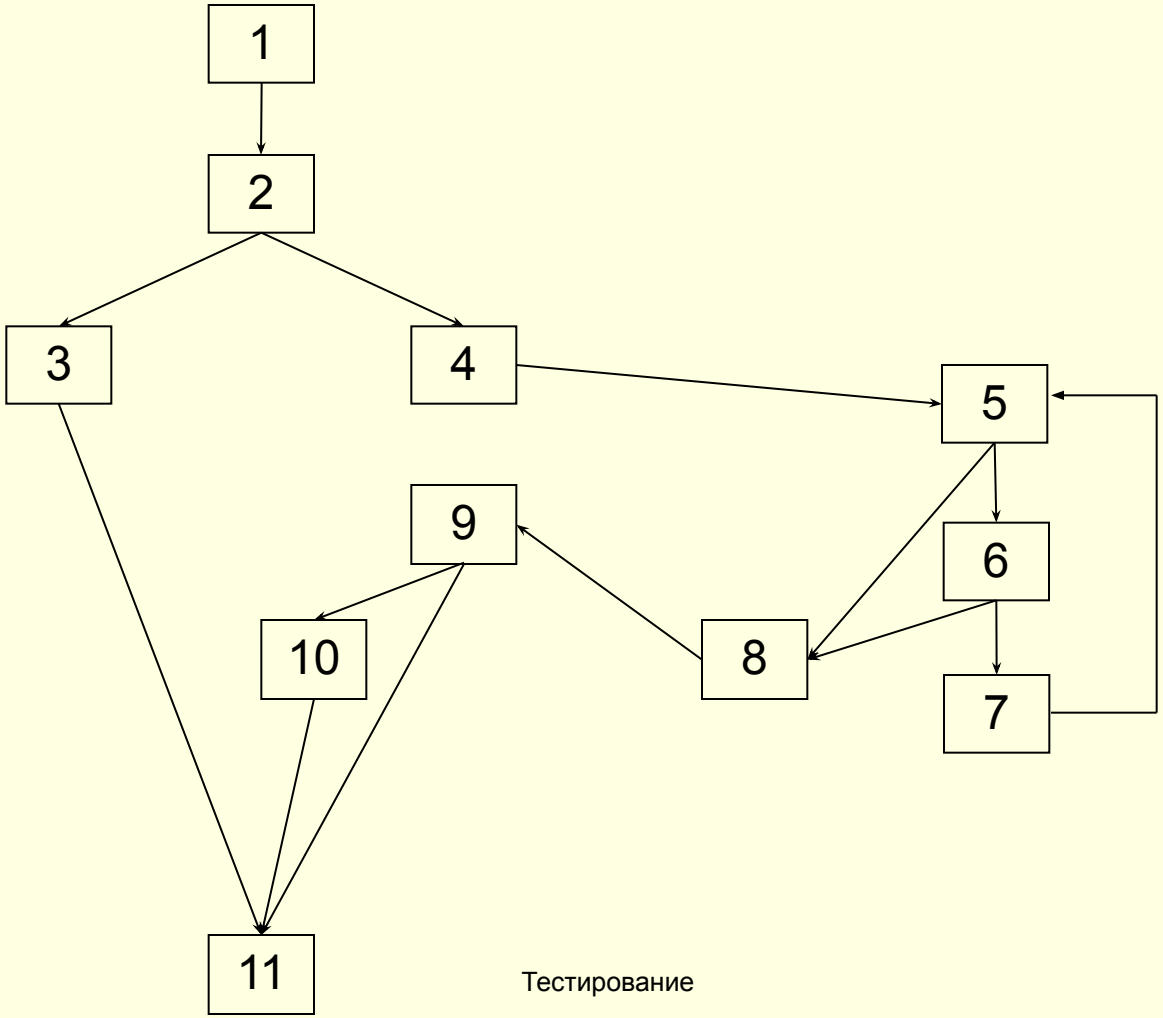
Пример

- Рассмотрим процедуру добавления элемента в упорядоченный линейный список
- Пронумеруем фрагменты исходного текста процедуры, которые будут соответствовать отдельным вершинам потокового графа
- Каждое из простых условий, входящих в составное, рассматривается как отдельный предикатный узел

Текст процедуры

```
void add (int val)
{      // создать новый элемент
  1  elem *p = new elem;  p->info = val;
  2  if (first == NULL)
     { // список пуст
  3     p->next = NULL;  first = p;  }
     else
     { // список  не пуст
   4    elem *q = first;
   5,6 while (q->next != NULL && q->info < val)
   7   q = q->next;
   8    p->next = q->next;   q->next = p; // вставить после указанного
   9    if (p->info < q->info)
  10    {      //  перестановка значений
        p->info = q->info;     q->info = val;   }
     }
  11 return;
}
```

Пример потокового графа



Базовое множество путей

- Множество независимых путей в потоковом графе, ведущих от начального узла к конечному, называется *базовым*
- Мощность этого множества называется его *цикломатической сложностью*
- Тестовый набор, обеспечивающий проверку всех путей базового множества, гарантирует хотя бы однократное выполнение каждого из операторов процедуры

Вычисление цикломатической сложности

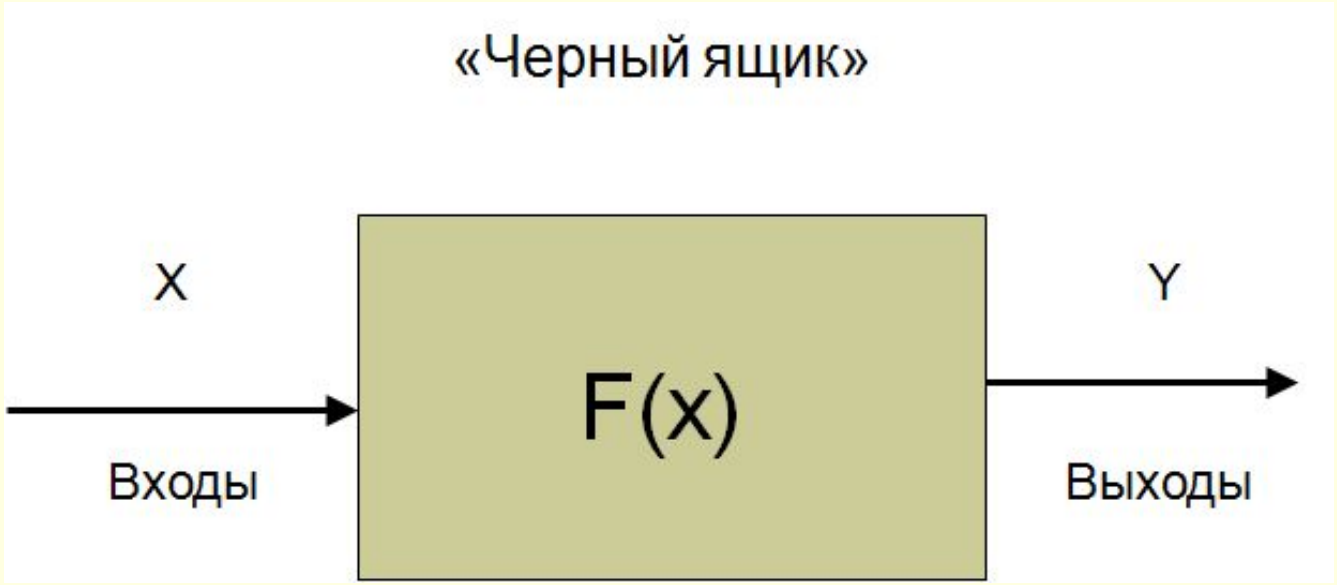
- Цикломатическую сложность можно определить одним из двух методов:
  - по формуле *E-V+2,* где E – число дуг, V – число узлов;
  - по формуле *p+1,* где p – число предикатных узлов
- Число тестовых вариантов, необходимых для полного покрытия равно цикломатической сложности

Итог

- Достоинства:
  - возможность предварительной оценки требуемого уровня тестового покрытия;
  - возможность учета особенностей программных ошибок;
  - высокая степень локализации ошибок
- Недостатки:
  - сложность подготовки тестовых наборов;
  - анализ результатов тестирования требует знания деталей реализации

ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ

Функциональное тестирование



Базируется на том, что структура тестируемого ПС неизвестна – тестирование по принципу «черного ящика»

Основное назначение

- Основное назначение функционального тестирования – проверка интерфейса ПС
- Функциональные тесты проверяют:
  - как выполняются функции программы
  - как принимаются исходные данные
  - как вырабатываются результаты
  - как сохраняется целостность внешней информации

Формирование тестов

- Тесты формируются, исходя только из функциональной спецификации программного средства
- Тестовое покрытие должно обеспечить проверку выполнения этой спецификации при различных комбинациях исходных данных

Формирование тестов

- Разработка функциональных тестов базируется на принципах:
  - на каждую используемую функцию или возможность − хотя бы один тест,
  - на каждую область и на каждую границу изменения какой-либо входной величины − хотя бы один тест,
  - на каждую особую (исключительную) ситуацию, указанную в спецификациях, − хотя бы один тест.

Формирование тестов

- Чаще всего используют два способа формирования тестовых наборов:
  - разбиение на классы эквивалентности,
  - анализ граничных значений
- Эти способы являются взаимодополняющими и могут применяться совместно

Классы эквивалентности

- Область исходных данных программы разбивается на *классы эквивалентности*
- Класс эквивалентности – это подмножество исходных данных, в пределах которого поведение программы одинаково
- Иначе говоря для любых двух наборов исходных данных из одного класса эквивалентности реализуется один и тот же базовый путь

Формирование классов

- Классы эквивалентности определяются по спецификациям входных данных в случаях, когда эти данные ограничены:
  - диапазоном значений (m..n);
  - множеством значений {a,b,c};
  - булевым множеством (true,false)
- В первом случае имеется три класса эквивалентности, во 2-м и 3-м – по два
- На каждый класс эквивалентности - тест

Анализ граничных значений

- Особенности данного способа:
  - тестовые варианты создаются только для границ областей эквивалентности;
  - при создании тестов учитываются не только условия ввода, но и условия вывода

Правила анализа

1. Если условия ввода задают непрерывный диапазон значений m..n, то тестовые варианты создаются для:
   - значений m и n,
   - значений m-ε и n+ε
2. Если условия ввода задают дискретный набор значений, то тестовые варианты создаются для:
   - проверки min и max значений,
   - проверки значений <min и >max

Правила анализа

3. Правила 1 и 2 применяются и к условиям вывода
4. Если внутренние структуры данных имеют предписанные границы, то создаются тесты, проверяющие эти структуры на их границах
5. Если входные и выходные данные суть упорядоченные множества, то тестируется обработка их первых и последних элементов

Пример

- Построить классы эквивалентности для процедуры бинарного поиска Key в M
- Предусловия:
  - M упорядочен;
  - M имеет не менее одного элемента;
  - нижняя граница <= верхняя граница
- Постусловия:
  - элемент найден – Result=True, I=номер;
  - элемент не найден – Result=False, I не определено;

Дерево разбиения

- Формирование классов эквивалентности выполняется с помощью *дерева разбиений,* листья которого дают искомые классы эквивалентности
- Последовательность построения дерева:
  1. проверка выполнения предусловий;
  2. проверка выполнения постусловий;
  3. анализ специальных требований;
  4. анализ граничных условий

Специальные требования

- Учитывают специфику выполнения конкретных алгоритмов обработки
- В нашем примере к числу специальных требований можно отнести следующие эквивалентные разбиения:
  - массив из одного элемента;
  - массив из четного числа элементов;
  - массив из нечетного числа элементов

Граничные условия

- Формулируются для узлов уровня специальных требований
- В нашем примере возможны следующие классы эквивалентности:
  - искомое значение хранится в первом элементе массива;
  - искомое значение хранится в последнем элементе массива;
  - искомое значение хранится в промежуточном элементе массива

Тестовые варианты

- В результате получается следующее дерево разбиения
- Это дерево имеет 11 листьев, каждый из которых задает отдельный тестовый вариант

Итог

- Достоинства:
  - независимость от реализации;
  - относительная простота подготовки тестов;
  - возможность анализа результатов специалистами предметной области
- Недостатки:
  - слабая локализация ошибок

Соотношение подходов

- Структурное и функциональное тестирование не альтернативные, а взаимодополняющие подходы
- Поэтому оптимальная стратегия проектирования тестов должна сочетать их в себе (тестирование «серого ящика»)
- Обычно на начальных стадиях тестирования применяют методы структурного тестирования, а на поздних – функционального

Стадии тестирования

- В процессе разработки программного средства обычно выделяют три стадии тестирования:
  - *модульное (компонентное),*
  - *интеграционное (комплексное),*
  - *системное (оценочное)*
- Эти стадии различаются как объемом тестируемой части ПС, так и уровнем диагностируемых ошибок

Характеристика этапов

- *Тестирование модулей.* Цель – индивидуальная проверка каждого модуля
- *Тестирование интеграции.* Цель – проверка межмодульных интерфейсов
- *Системное тестирование.* Цель – проверка выполнения всех требований к ПС

Модульное тестирование

- *Модульному тестированию* подвергаются небольшие модули (процедуры, классы и т. п.)
- Тестирование осуществляется по методу «белого ящика» и проверке подвергаются:
  - интерфейс модуля;
  - внутренние структуры данных;
  - независимые пути выполнения;
  - граничные условия;
  - пути обработки ошибок

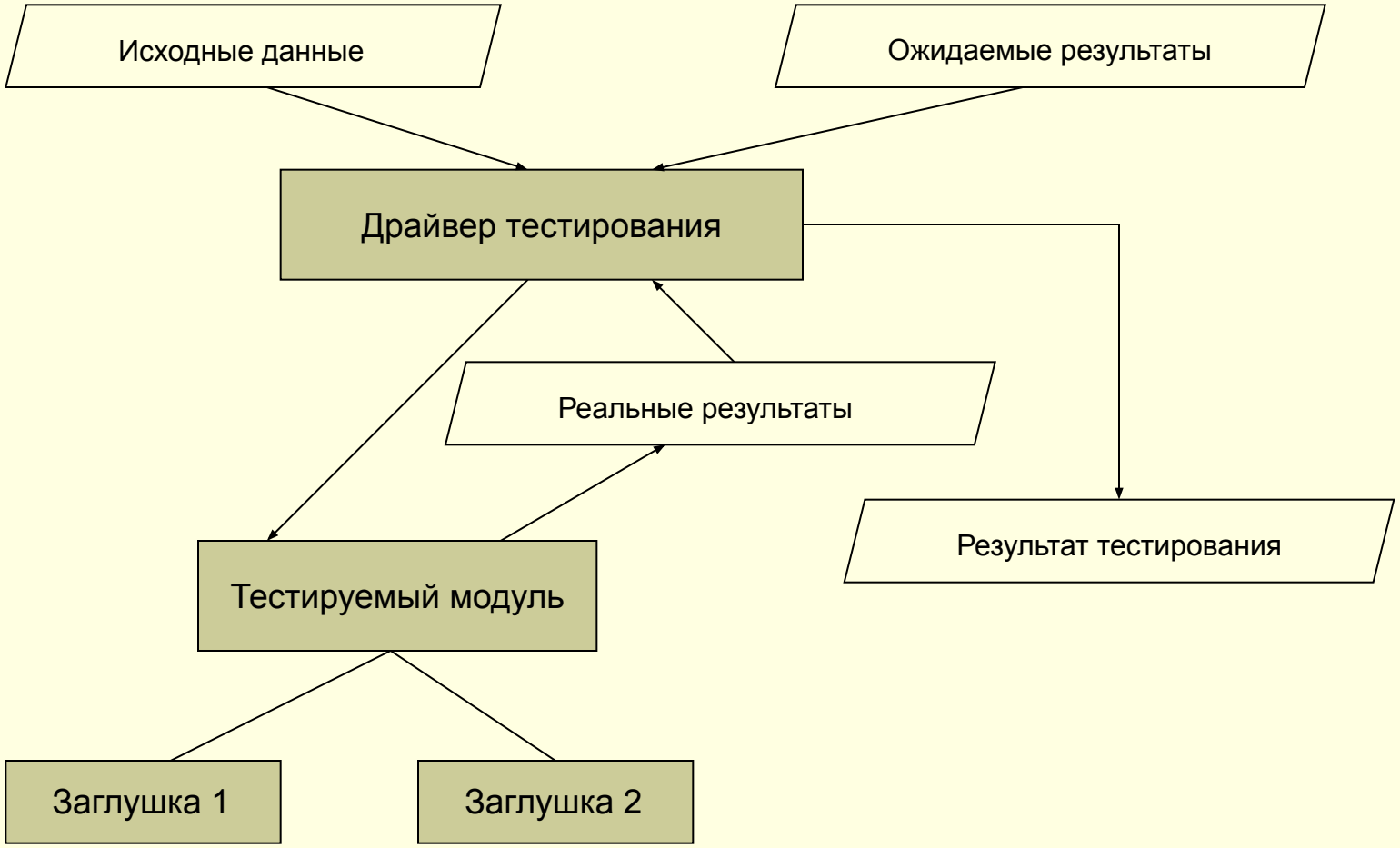
Модульное тестирование

- Модульное тестирование обычно рассматривается как дополнение к этапу кодирования
- Модуль не является автономной системой, поэтому его тестирование требует использования дополнительных средств:
  - драйверов тестирования,
  - заглушек

Драйверы и заглушки

- Драйвер – это управляющая программа, которая:
  - принимает исходные данные и ожидаемые результаты тестов,
  - вызывает тестируемый модуль,
  - преобразует полученные от него реальные результаты в удобную для анализа форму
- Заглушка – это процедура, реализующая интерфейс замещаемого модуля и, возможно, выполняющая минимальную обработку данных

Среда для тестирования модуля



Интеграционное тестирование

- *Интеграционное тестирование* – это отладочное тестирование постепенно наращиваемой системы
- Система строится поэтапно путем добавления отдельных модулей и их групп
- На каждом этапе после приращения системы производится ее тестирование

Компонентное тестирование

- Распространение компонентных технологий породило термин *компонентное тестирование* как частный случай интеграционного тестирования
- В этом случае тестированию подвергаются *компоненты* - обладающие определенной функциональностью и готовые к использованию фрагменты кода

Методы тестирования

- Интеграция системы может осуществляться в направлении сверху - вниз или снизу - вверх
- Соответственно, различают два метода тестирования, поддерживающих процесс интеграции:
  - *нисходящее тестирование интеграции,*
  - *восходящее тестирование интеграции*

Нисходящее тестирование

- При нисходящем тестировании первым тестируется головной модуль программы, который представляет всю тестируемую программу
- Он тестируется при «естественном» состоянии информационной среды, при котором начинает выполняться эта программа

Нисходящее тестирование

- Те модули, к которым может обращаться головной, заменяются их отладочными имитаторами (*заглушками*)
- Затем одна из заглушек заменяется реальным модулем и выполняется набор тестов, проверяющих эту структуру
- Процесс подключения продолжается вплоть до получения нужной конфигурации

Характеристика нисходящего тестирования

- Достоинство: Ошибки в главной, управляющей части системы выявляются в первую очередь
- Недостаток: Трудности в ситуациях, когда для полного тестирования на верхних уровнях нужны результаты, полученные на нижних уровнях

Восходящее тестирование

- Модули нижнего уровня объединяются в несколько кластеров, каждый из которых выполняет определенную подфункцию
- Для каждого кластера создается программу-драйвер
- Тестируется кластер
- Драйвер удаляется, а кластеры объединяются в структуру движением вверх

Характеристика восходящего тестирования

- Достоинство: Простота подготовки тестов, отсутствие заглушек
- Недостаток: Система не существует как целое, пока не будет добавлен последний модуль

Системное тестирование

- Полностью реализованный программный продукт подвергается *системному тестированию*
- На этом этапе тестировщика интересует программная система в целом, как ее видит конечный пользователь
- Основой для тестов служат общие требования к системе – корректность реализации функций, производительность, время отклика, устойчивость к сбоям и т.д.

Системное тестирование

- Основные виды системных тестов:
  - функциональное тестирование (по методу «черного ящика»),
  - тестирование восстановления,
  - тестирование безопасности,
  - стрессовое тестирование,
  - тестирование производительности

Критерии тестового покрытия

- Для системного и компонентного тестирования используются специфические виды критериев тестового покрытия:
  - тестирование всех типовых сценариев работы;
  - тестирование всех сценариев с нештатными ситуациями;
  - тестирование попарных композиций сценариев и т.д.

Альфа-тестирование

- Данная стадия включает тестирование системы конечным пользователем, так называемое альфа- и бета-тестирование
- *Альфа-тестирование* - тестирование проводимое заказчиком в организации разработчика
- Разработчик фиксирует все выявленные ошибки и недостатки использования

Бета-тестирование

- *Бета-тестирование* - опробование программного продукта потенциальными пользователями на реальных задачах
- О найденных ошибках и замечаниях пользователь сообщают разработчику
- Тестируемая таким образом версия программного средства называется *бета-версией* и, как правило, она предшествует коммерческому выпуску продукта

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ТЕСТИРОВАНИЕ

Отличия от классического

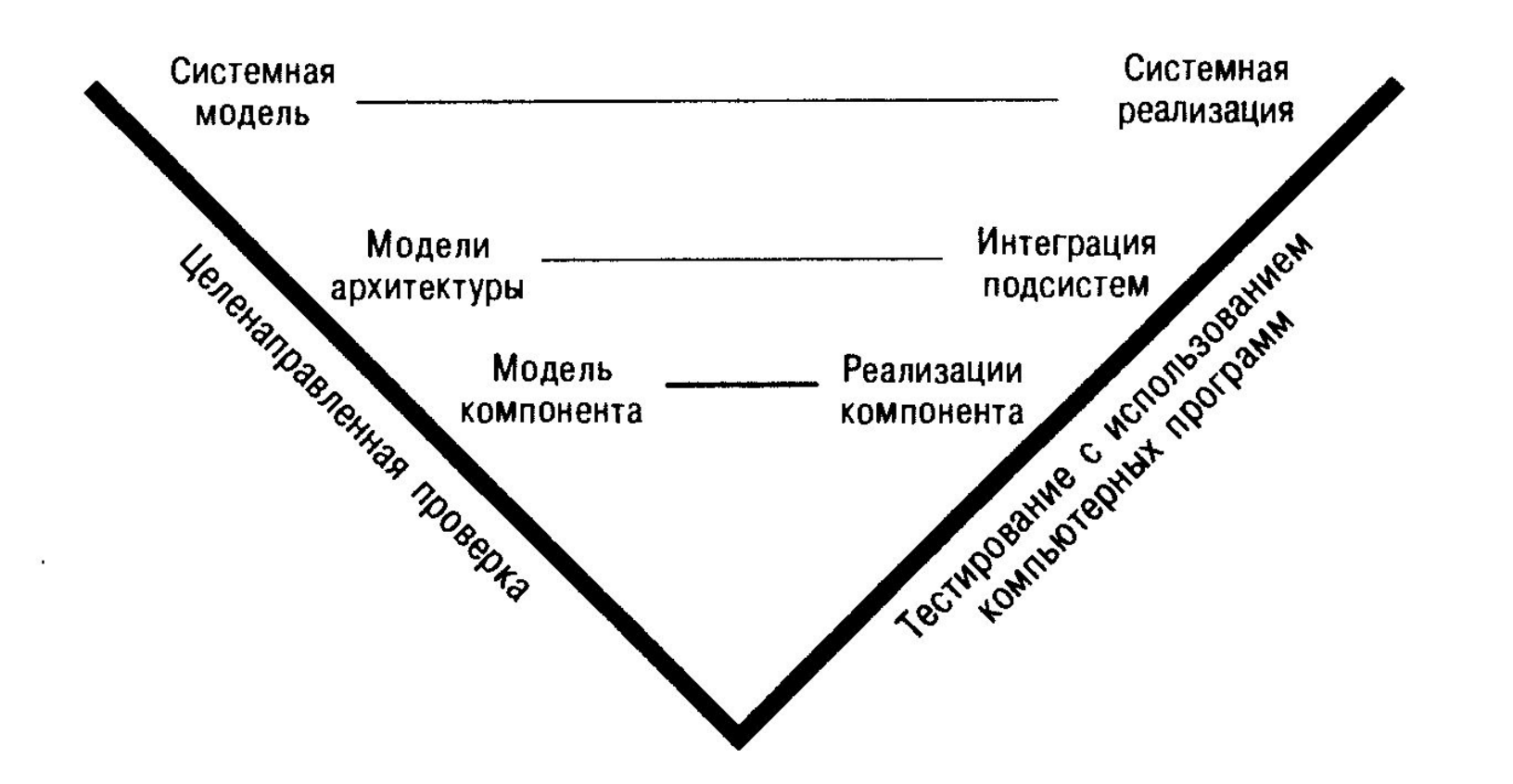
- Тестирование объектно-ориентированных программных средств имеет ряд существенных отличий от классического тестирования:
  - расширение области применения тестирования;
  - изменение методики тестирования;
  - учет особенностей ООП при проектировании тестовых вариантов

Расширение области применения

- Разработка объектно-ориентированного программного средства начинается с создания его визуальных моделей
- Модели этапа анализа и этапа проектирования определяют основные функциональные и структурные свойства разрабатываемой системы, поэтому

Необходимо проводить тестирование этих моделей !

V-образная модель тестирования



Критерии тестирования моделей

- Модели разрабатываемой системы должны удовлетворять критериям:
  - синтаксической и семантической правильности,
  - полноты,
  - согласованности

Правильность модели

- Синтаксическая правильность связана с корректным использованием нотаций языка описания моделей
- Семантическая правильность определяется соответствием модели реальной системе и связанной с ней задаче
- Тестирование подтверждает, что модель правильна в отношении конкретного тестового случая, если результат его выполнения является ожидаемым.

Полнота модели

- Мера наличия в модели необходимых элементов
- Тестирование показывает, существуют ли сценарии, которые не могут быть представлены элементами, входящими в состав модели
- Модель считается полной, если результаты выполнения тестовых случаев могут быть адекватно представлены содержимым самой модели

Согласованность модели

- Мера присутствия противоречий внутри модели или между текущей моделью и моделью, на базе которой она была построена
- Тестирование выявляет такие противоречия, находя в модели различные представления подобных тестовых случаев

Изменение методики тестирования

- Как и для процедурных, для объектно-ориентированных программных систем выделяют три стадии тестирования:
  - *модульное (компонентное),*
  - *интеграционное (комплексное),*
  - *системное (оценочное)*
- Изменение методики тестирования касается, в основном, двух первых стадий

Модульное тестирование

- Наименьшим тестируемым элементом объектно-ориентированного ПО является не процедура, а *класс*
- Поскольку класс содержит набор свойств и методов, образующих единую сущность, изолированное тестирование методов не имеет смысла
- Методы должны тестироваться в контексте частных свойств и операций класса

Тестирование классов

- Автономное тестирование класса предполагает разработку драйвера, который будет:
  - создавать экземпляры тестируемого класса;
  - вызывать методы тестируемого класса и передавать им фактические параметры из тестовых вариантов;
  - принимать результаты выполнения тестируемых методов

Тестовый драйвер

- Существует несколько способов реализации тестового драйвера:
  - в виде отдельного класса – тестирование public-части класса;
  - в виде класса, наследуемого от тестируемого – тестирование protected-части;
  - в виде статического метода внутри тестируемого класса – тестирование private-части

Тестирующий класс

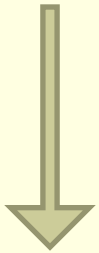
- Методы этого класса создают объекты тестируемого класса и вызывают их методы, в том числе и статические
- Преимущества:
  - возможность многократного использования драйвера при тестировании классов-наследников;
  - достижение максимальной компактности и быстродействия рабочего кода

Тестирующий метод

- Преимущества:
  - непосредственная близость программного кода драйвера к программному коду тестируемого класса;
  - возможность многократного использования кода драйвера (в силу наследования) для тестирования классов-наследников
- Недостаток:
  - необходимость отделения программного кода драйвера от поставляемого ПО

Тестирование классов

- Экземпляры отдельных классов в активно взаимодействуют между собой



- Создание драйвера для автономного тестирования класса может оказаться не менее сложной задачей, чем разработка самого класса

Тестирование классов

- Решение об автономном тестировании класса принимается с учетом следующих факторов:
  - роли класса в системе;
  - сложности класса, измеряемой числом состояний, операций и связей с другими классами;
  - объема трудозатрат, связанных с разработкой тестового драйвера

Роль класса

- Роль класса в разрабатываемой системе тем выше, чем больше связанные с ним риски
- Выделение таких *базовых* классов возможно на основе тщательного анализа проблемы и только после определения множества классов

Сложность класса

- С точки зрения взаимодействия можно выделить два типа классов:
  - примитивные классы;
  - непримитивные классы
- Экземпляры *примитивного* класса можно использовать без необходимости создания экземпляров каких-либо других классов, в том числе и данного класса
- Такие объекты представляют собой простейшие компоненты системы

Сложность класса

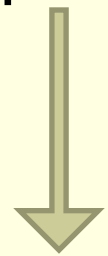
- Число примитивных классов в системе обычно невелико
- Основная роль в объектно-ориентированных системах отводится *непримитивным* классам
- Объекты непримитивных классов требуют использования других объектов при выполнении своих операций

Сложность создания драйвера

- Трудоемкость создания тестового драйвера тем выше, чем выше степень его связности с другими классами
- Тем не менее, он должен удовлетворять следующим требованиям:
  - иметь сравнительно простую структуру;
  - быть удобным в сопровождении;
  - легко модифицируемым в ответ на изменения в спецификации тестируемого класса

Тестирование интеграции

- Объектно-ориентированное ПО не имеет иерархической управляющей структуры



- Методики нисходящего и восходящего тестирования здесь неприменимы
- Зачастую неосуществим классический прием интеграции – добавление по одной операции в класс

Тестирование интеграции

- Основная цель этого этапа тестирования – проверка правильности обмена сообщениями между объектами, классы которых уже прошли тестирование в автономном режиме
- Основная задача – выделение подмножества взаимодействующих классов

Виды взаимодействия классов

1. Метод одного класса содержит в списке своих формальных параметров имена других классов
2. Метод одного класса создает экземпляр другого класса как часть своей реализации
3. Метод одного класса ссылается на глобальный экземпляр другого класса

Тестирование интеграции

- Наиболее популярными являются следующие методики тестирования интеграции объектно-ориентированных систем:
  - тестирование, основанное на потоках;
  - кластерное тестирование

Тестирование потоков

- Объектом интеграции является набор классов, обслуживающих единичный ввод данных в систему
- При наличии в системе нескольких потоков ввода средства обслуживания каждого из них тестируются отдельно
- Для контроля побочных эффектов применяют регрессионное тестирование

Кластерное тестирование

- Объектом тестирования является *кластер* – набор сотрудничающих классов
- Для выделения кластеров можно использовать диаграммы взаимодействия, соответствующие отдельным прецедентам

Размер кластера

- При малых размерах кластера невозможно воспроизведение в полном объеме эффекта интеграции (системного эффекта)
- Однако, с увеличением размера кластера возрастает вероятность возникновения не фиксируемых тестами ошибочных промежуточных результатов

Среда тестирования

- Тестирование кластеров можно проводить:
  - непосредственно в среде приложения;
  - в среде, специально созданной тестирующим драйвером
- В первом случае:
  - требуется выделять результаты тестирования из общих информационных потоков в программной системе;
  - результаты тестирования соответствуют реальным условиям эксплуатации

Среда тестирования

- Во втором случае:
  - результаты тестирования получаются в «чистом» виде;
  - соответствие результатов тестирования реальным условиям эксплуатации зависит от степени адекватности этим условиям созданной драйвером среды тестирования

Системное тестирование

- В основном его методика совпадает с методикой классического тестирования

Конец лекции