Понятие тестирования ПО. Основные определения

Тести́рование програ́ммного обеспе́че́ния — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом

Mistake (Error) - Ошибка. просчет. (человека)
Fault - дефект, изьян (ПО в результате ошибки)
Failure - Неисправность, отказ, сбой. (Внешнее проявление дефекта)
Error - Невозможность выполнить задачу вследствие отказа.

Цели тестирования. Классификация тестов

- Цели тестирования:
  - Обнаружение дефектов
  - Повышение уверенности в уровне качества
  - Предоставление информации для принятия решений
  - Предотвращение дефектов

Увеличение приемлемого уровня пользовательского доверия в том, что программа функционирует корректно во всех необходимых обстоятельствах

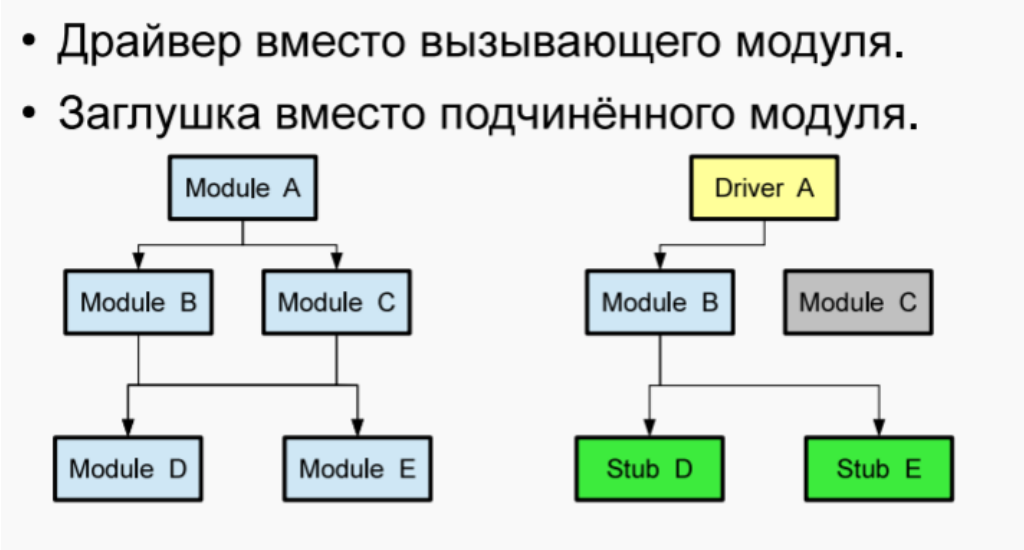
- Уровень доверия
- Корректное поведение
- Необходимые обстоятельства - требование реального окружения

Модульное тестирование. Понятие модуля

Модульное (компонентное) тестирование - тестирование отдельных компонентов ПО.
Модуль - это компонент, который необходимо протестировать отдельно от остального программного продукта. Модуль выполняет некую законченную функцию. Они определены в дизайне программы. (Разделение сначала на слои, затем на модули). Для проведения модульного тестирования, модуль необходимо изолировать из системы.

Изолирование модулей

Изолирование модулей производится ради исключения сторонних воздействий.



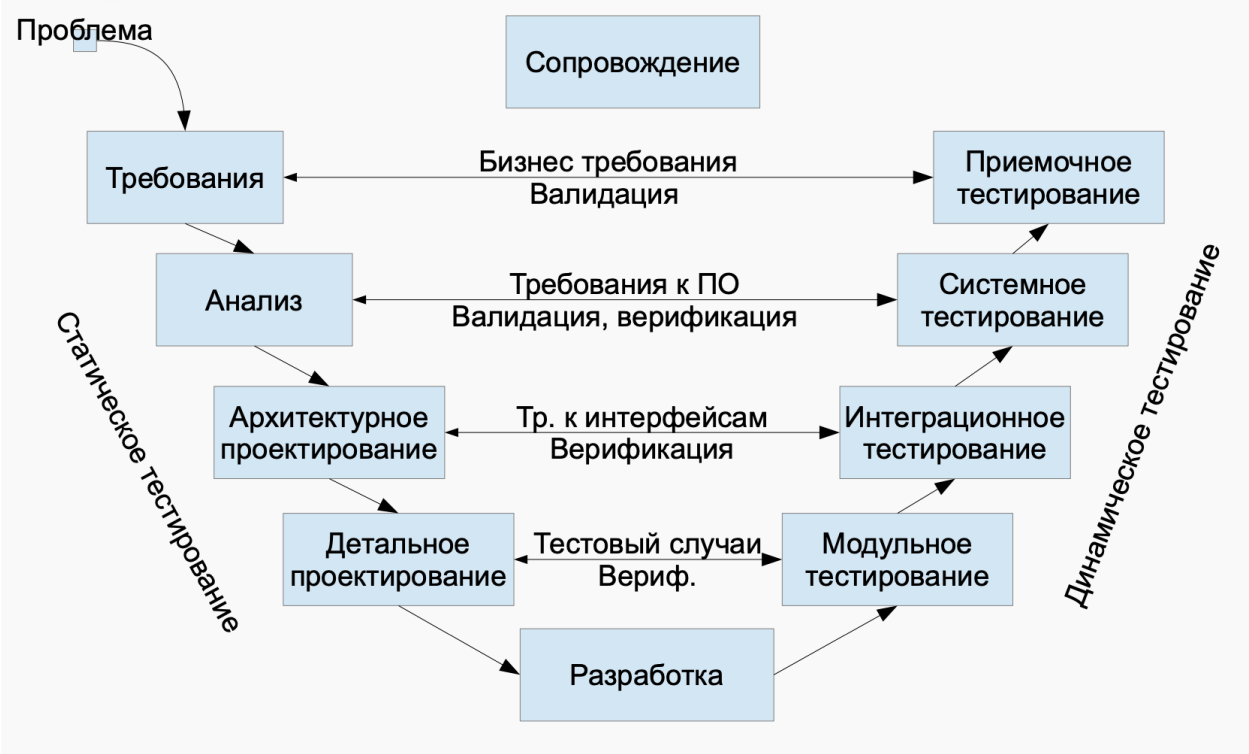
> Программа, представленная на слайде, состоит из 5 модулей. При этом главным из них является модуль верхнего уровня, который осуществляет вызовы всех остальных. Модули верхнего уровня зависят от модулей, которые расположены ниже по иерархии вызовов. Изолирование модулей предполагает замену модулей, осуществляющих вызов *драйверами* (т.е. управляющими тестированием), а подчинённых модулей — *заглушками*. Иногда для этого требуется отдельная тестовая сборка приложения, в которой присутствует код только для одного модуля. При этом количество таких сборок обычно равно количество модулей в системе.

Драйвер - компонент, вызывающий модули и обеспечивающий последовательность тестирования. Он должен последовательно вызывать тестируемый модуль с различными входными параметрами и условиями.

Заглушка ведет себя подобно подчиненному модулю, имеет тот же интерфейс, но гораздо более простую реализацию. При вызове заглушка возвращает определенное заранее значение.

Для модульного тестирования в Java используется JUnit. Это простейший фреймворк, позволяющий создать модульные тесты и выполнить их в определенном окружении.

V-образная модель. Статическое и динамическое тестирование



Статическое (рецензирование)
– Не включает выполнения кода
– Ручное, автоматизированное
– Неформальное, сквозной контроль, инспекция
Динамическое
– Запуск модулей, групп модулей, всей системы
– После появления первого кода (а иногда перед!)

Валидация и верификация. Тестирование методом "чёрного" и "белого" ящика

Валидация
– Проверка на соответствие ожиданиями
– ПО выполняет требования пользователя?
– Пирожок (мясной, вегетарианский, сладкий)
– Have we done the right thing?
Верификация
– Внутреннее управление качеством
– ПО выполняет требования спецификации?

– Пирожок (размер, степень прожарки, начинка, …)
– Have we done the thing right?

«Белый ящик» – когда у нас есть доступ к коду, и мы его тестируем, читаем сам код (статическое тестирование), запускаем в дебаге, пишем автотесты;

«Черный ящик» – когда мы не знаем, как система устроена внутри, нет доступа к коду или мы не умеем его читать, и поэтому ориентируемся только на внешнее поведение или ТЗ;

«Серый ящик» – когда мы смотрим в код и понимаем, как он устроен, а потом открываем само приложение и проверяем, как этот код отображается уже в нем, но ориентируемся уже больше на ТЗ (это компиляция двух вышеприведенных определений).

Тестовый случай, тестовый сценарий и тестовое покрытие

- Входные значение
  - Данные или управляющие воздействия
- Предусловия, условия выполнения, постусловия
- Ожидаемый результат
  - Выходные данные и состояния, изменения в них, и другие последствия теста
  - Определен до запуска теста! (в идеале и TDD)
- Повторяемый, автоматизируемый
- Учитывает состояния (если есть)
  - Переходы между состояниями
- Правильные: корректный результат
- Неправильные : корректные сообщения об ошибках
- В российской официальной терминологии используется термин сценарий

Последовательность случаев – Типичное использование системы

№	Начальное состояние	Ввод	Действие системы	Вывод	Конечное состояние
1	Готов	Пользователь вставляет карточку	Успешное чтение карточки	Приглашение “введите pin”	Ожидание pin-кода
2	Ожидание pin-кода	Вводим **верный** pin-код	Проверка pin-кода	Приглашение к выбору транзакции	Ожидание выбора транзацкии
3	Ожидание выбора транзацкии	Выбор выдачи 5000 рублей	Проверка баланса, возможности выдачи	Деньги	Выдача денег
4	Выдача денег	Пользователь берет деньги и карточку	Завершение выдачи	Благодарность за использование	Готов

Должны обрабатывать – Корректное поведение и вариант ошибки

№	Начальное состояние	Ввод	Действие системы	Вывод	Конечное состояние
1	Готов	Пользователь вставляет карточку	Успешное чтение карточки	Приглашение “введите pin”	Ожидание pin-кода
2	Ожидание pin-кода	Вводим **неверный** pin-код	Проверка pin-кода	Сообщение об ошибке	Ожидание pin-кода
3	Ожидание pin-кода	Вводим **неверный** pin-код	Проверка pin-кода	Сообщение об ошибке	Ожидание pin-кода
4	Ожидание pin-кода	Вводим **неверный!!!** pin-код	Проверка pin-кода, блокировка карточки	Сообщение об ошибке и блокировка	Готов

- Определение корректного поведения в:
  - Требованиях (системное, приемочное тестирование)
  - Архитектуре (интеграционное тестирование)
  - Проектных документах (модульное тестирование)
- Тестовые сценарии
  - Можно взять из документации к проекту: Use-case → Test-case

Тестовое покрытие:

- Требуется баланс – Много тестов → больше покрытие → качество выше – Меньше тестов → выше скорость разработки → быстрее выход на рынок
- Необходимо выбрать специфические значения для тестирования
  - Нельзя же тестировать вечность!
  - Полное покрытие недостижимо

Выбор тестового покрытия:

- Эквивалентное разбиение (партиции эквивалентности)
    - – Анализ граничных значений
- Таблица решений (альтернатив)
- Таблицы переходов
- Сценарии использования

Тестовый случай состоит из набора входных значений, предусловий выполнения, ожидаемых результатов и постусловий.

Содержание спецификаций проектирования тестов (включая тестовые условия) и спецификаций тестовых сценариев описывается в стандарте «Документация при тестировании программ» (IEEE STD 829-1998).

Ожидаемые результаты должны создаваться как часть спецификаций тестовых сценариев и включать в себя выходные данные, изменения в данных и состояниях, и любые иные последствия теста. Если ожидаемые результаты не были определены, правдоподобные, но ошибочные результаты могут быть приняты за корректные.

В идеальных условиях ожидаемые результаты должны быть определены до момента выполнения теста. Для отражения этой концепции был разработан подход Test Driven Development — разработка через тестирование. При этом подходе разработчик сначала описывает тестовое покрытие и разрабатывает тесты для этого покрытия. После этого он начинает разрабатывать собственно программное обеспечение, и с каждым его запуском растет количество тестов, выполненных успешно. Разработка считается завершённой, когда будут успешно выполнены все тесты.

Тестовый случай должен быть повторяемым (одинаковый набор входных значений и состояний должен приводить к одинаковым входным значениям или состояниям).
Для повторяемости теста желательно автоматизировать его поведение.
В тестовом случае должны учитываться состояния внутри ПО (если они есть) и переходы между ними -> расширение тестового покрытия.
Необходимо тестировать и нормальные сценарии, и сценарии, приводящие к появлению сообщений об ошибках.
Тестовый сценарий - это последовательность тестовых случаев.

При планировании тестирования следует соблюдать баланс между качеством и скоростью вывода продукта в эксплуатацию. Количество тестовых сценариев необходимо выбирать с учетом того, что полное тестовое покрытие недостижимо.

Анализ эквивалентности

- Эквивалентное разбиение (партиции эквивалентности) - анализ граничных значений, внутри которых тестируемая функция ведет себя одинаково

При анализе эквивалентности тестируемая функция или модуль разбивается на участки, где программа ведет себя одинаково (эквивалентно). Внутри каждого участка формируется свой набор тестовых случаев. Если таких участков относительно немного, это позволяет резко сократить количество тестовых случаев. Отдельные тесты составляются для граничных значений участков.

Таблицы решений и таблицы переходов.

Decision Table (таблица решений) — техника, помогающая наглядно изобразить комбинаторику условий из ТЗ.

- По горизонтали — выписываем условия, которые влияют на результат. А чуть ниже — сам результат, в оригинале Action — действие, которое нужно выполнить.
- По вертикали — правила: конкретная комбинация входных условий.

То есть мы указываем значения условий и результата

	Правило 1	**Правило 2**	**...**	**Правило N**
Условия				
Условие 1				
Условие 1				
...				
Условие N				
Действия				
Действие 1				
Действие 2				

таблицы переходов – это метод тестирования «черного ящика», который используется там, где некоторый аспект системы может быть описан в так называемом «конечном автомате». Это просто означает, что система может находиться в (конечном) числе разных состояний, а переходы из одного состояния в другое определяются правилами «машины».

Регрессионное тестирование

Регрессионное тестирование – это набор тестов, направленных на обнаружение дефектов в уже протестированных участках приложения. Делается это совсем не для того, чтобы окончательно убедиться в отсутствии багов, а для поиска и исправления регрессионных ошибок. Регрессионные ошибки – те же баги, но появляются они не при написании программы, а при добавлении в

существующий билд нового участка программы или исправлении других багов, что и стало причиной возникновения новых дефектов в уже протестированном продукте.

Библиотека JUnit. Особенности API. Класс junit.framework.Assert.

JUnit — библиотека для модульного тестирования программ Java.

API: Самый важный пакет в JUnit — это **junit.framework** , который содержит все основные классы. Некоторые из важных классов следующие:

Sr.No.	Имя класса	функциональность
1	утверждать	Набор методов assert.
2	Прецедент	Тестовый пример определяет прибор для запуска нескольких тестов.
3	Результат испытаний	TestResult собирает результаты выполнения контрольного примера.
4	Тестирование	TestSuite — это набор тестов.

Assert Class

Ниже приводится объявление для класса **org.junit.Assert** —

```
public class Assert extends java.lang.Object
```

Этот класс предоставляет набор методов утверждения, полезных для написания тестов. Только ошибочные утверждения записываются. Вот некоторые из важных методов класса Assert:

Отличия JUnit 3 от JUnit 4

1. Аннотации Java 5 для установки и демонтажа (@before и @after) вместо setUp() и tearDown().
2. больше не нужно расширять класс TestCase .
3. @Test аннотация заменяет testSomeMethod() соглашение об именах.
4. static imports для утверждений (то есть мы можем использовать методы вне нашего метода, а просто в классе).
5. Теории Junit, которые позволяют отделить наборы данных от самого теста.