

# Тестирование ПО

Клименков С.В.  
2020-2021 уч. Год  
v.1.11 от 07.09.2020

# Контактная информация

- Технические вопросы  
[https://vk.com/serge\\_klimenkov](https://vk.com/serge_klimenkov)
- Лекции  
<https://youtube.com/c/SergeKlimenkov>
- Материалы по курсу  
<https://se.ifmo.ru/>
- Комната 374
- Техническая беседа в ВК
- ~~ИСУ, Электронная почта~~

# Литература

- Иан Соммервилл. Инженерия программного обеспечения
- Рекс Блэк «Advanced Software Testing»,
- Рекс Блэк Ключевые процессы тестирования.
- ISTQB - International Software Testing Qualifications Board
  - [www.istqb.org](http://www.istqb.org)
  - [www.rstqb.org](http://www.rstqb.org)
- P.Ammann, J.Offutt  
Introduction to Software Testing
  - [www.cs.gmu.edu/~offutt/softwaretest/](http://www.cs.gmu.edu/~offutt/softwaretest/)

# Основы тестирования

1

# Системы с программным обеспечением

- Бизнес-системы
- Аппаратура с ПО
- Потребительские товары
- Военные, космические системы
- Информационно-управляющие системы
- ...

Сильное давление со стороны бизнеса и гибких методологий

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

Отказ м.б. следствием  
окружающей среды

# Пример программы

```
public int countPositive (int [ ] data) {  
    int count = 0;  
    for (int i = 1; i < data.length; i++) {  
        if (data [ i ] > 0)  
            count++;  
    }  
    return count;  
}
```

Сколько здесь дефектов  
и к чему они могут привести?

# BUG

- Используется неформально
- Может обозначать
  - Дефект (fault)
  - Отказ (failure)
  - Невозможностью выполнить задачу (error)
  - Что то другое или ничего не обозначать



# Топ-6 катастроф по вине ПО:

- Космос: Ariane 5  
Ошибка модуля управления
- Деньги: Knight Capital  
500 миллионов за полчаса
- Медицина: Лучевая терапия  
Смерть из-за большой дозы радиации
- Интернет: Amazon  
Каскадный сбой, начавшийся с грозы
- Инфраструктура: «блэкаут» на северо-востоке США  
Сбой ПО мониторинга
- Транспорт: American Airlines  
Проблема в интеграции систем бронирования

# Уровни восприятия тестирования

- Уровень 0 – тестирование == отладка
  - Не отличает некорректное поведение и ошибки в программе
  - Не учитывает требования надежности и безопасности
- Уровень 1 - предназначение – показать корректность ПО
  - Невозможно доказать
  - Что значит “ошибок нет”?
  - Нет формальных правил

HW engineer

“Тестирование может  
показать наличие дефектов,  
а не их отсутствие”  
Edgar Dijkstra

# Уровни восприятия тестирования

- Уровень 2 - Поиск ошибок разработчиков
  - Конфликт разработчиков и тестировщиков
- Уровень 3 - Тестирование может показать наличие ошибок
  - Используя ПО мы подвержены рискам
  - Риск – последствия незначительные
  - **Риск** – последствия катастрофические
  - Тестировщики и разработчики **совместно** снижают риски

SW компании

«просветленные»  
SW компании

# Уровни восприятия тестирования.

## Тестирование и качество ПО

- Уровень 4 — Тестирование - это возможный способ оценки качества программного обеспечения в терминах найденных дефектов
  - Функциональное
  - Нефункциональное :надежность, практичность, эффективность, сопровождаемость и переносимость
- ISO 9126 «Информационная технология. Оценка программного продукта»
- ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE)

Традиционные  
производства

- Другие способы оценки качества
  - Разработка стандартов
  - Обучение
  - Анализ дефектов

# Цели тестирования (ISTQB)

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

# Цели тестирования

- Увеличение приемлемого уровня пользовательского доверия в том, что программа функционирует корректно во всех необходимых обстоятельствах
  - Уровень доверия
  - Корректное поведение
  - Необходимые обстоятельства - требование реального окружения

# Уровень доверия

- Наглядность
- Уровень остаточного обнаружения дефектов
  - Число дефектов обнаруженных тестом или набором тестов
  - Число дефектов обнаруженных в заданное время

«Меньше 10-ти критических дефектов найдено за последние 7 дней»

- Требования к надежности
  - Сложно показать без испытаний, т. е. работающего ПО

Среднее время между отказами не должно быть меньше 5000 часов



# Корректное поведение

- Необходимо определение
  - Из требований
  - Спецификаций, описаний, ...
  - Зависит от уровня тестирования

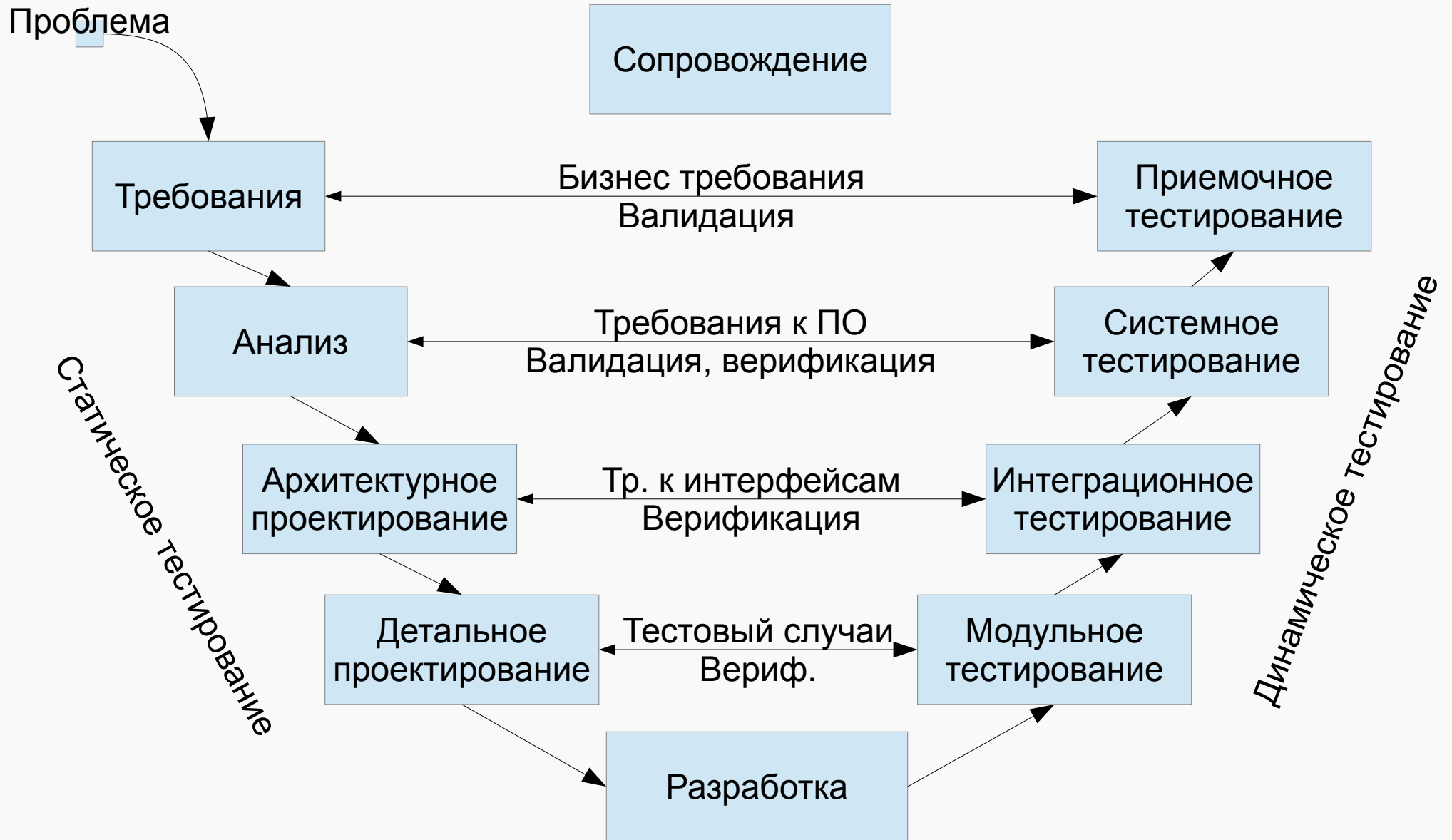
# Реальное окружение

- Реалистичное количество данных - таких же как в целевой системе
  - В университете 5000 студентов, небольшой рост
  - Необходим тест на 5000, 6000, 7000 студентов, но не на 100000
- Реалистичный набор, комбинация входных данных

# Полное тестовое покрытие

- `public long multiply (int A, int B)`
  - Как протестировать?
  - Сколько потребует памяти?
  - Сколько будет выполняться на 3ГГц ЦПУ?

# Традиционная V-model



# 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

- 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?

# Источники данных для тестов

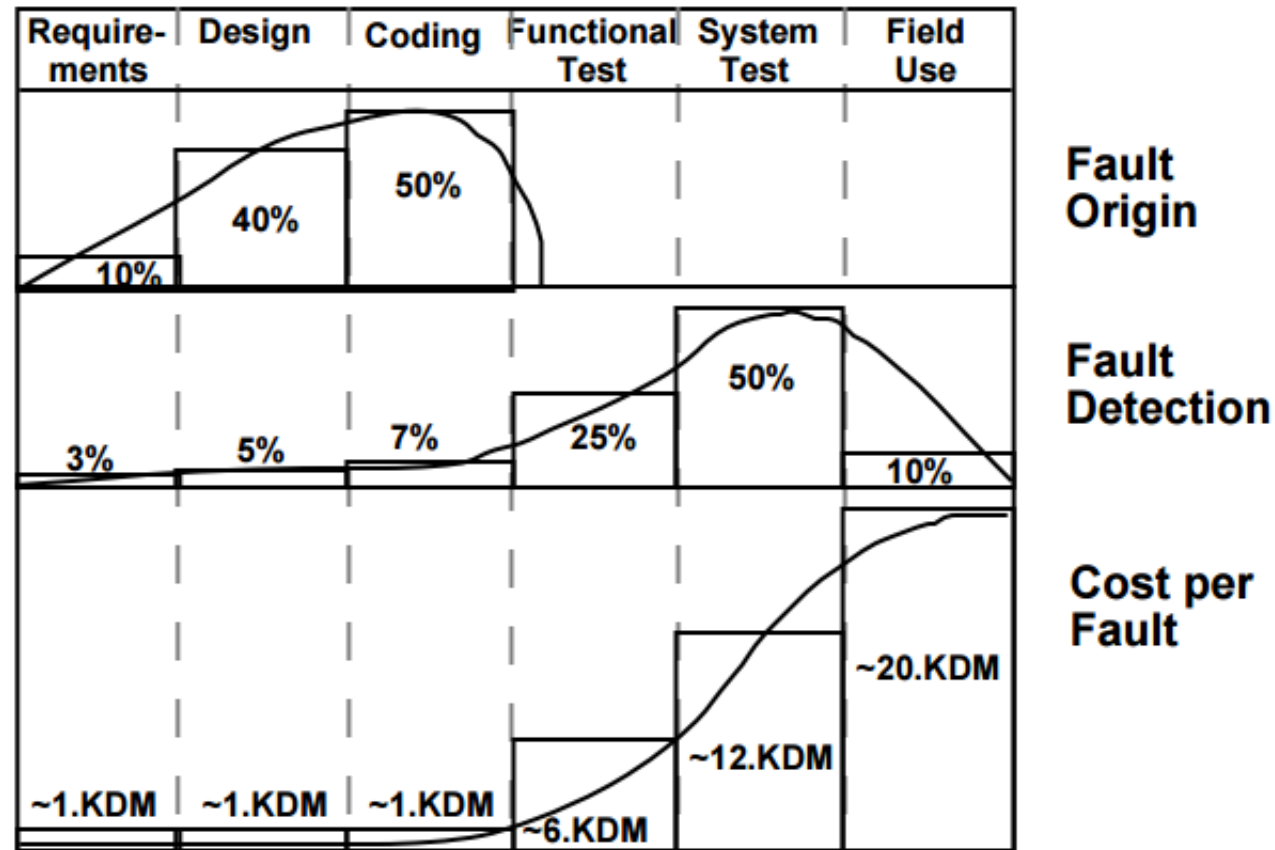
- Описания ПО — метод «черного ящика»
  - Спецификации, требования, дизайн.
  - Запуск и сравнение результатов с эталоном
- Исходный код — метод «белого ящика»
  - Переходы, утверждения, условия...
  - Анализ путей, структуры
- Опыт
- Модели
  - UML

# Деятельность и роли в тестировании

- Проектирование тестов
  - На основании формальных критериев
  - На основании знаний предметной области, опыта и экспертизы
- Автоматизация тестов
  - Знание средств, скриптов
- Исполнение тестов
  - Нет специальных требований к квалификации
- Анализ результатов
  - Знания предметной области



# Цена дефекта



KDM=kilo-deutsch marks



# Принципы тестирования (ISTQB)

1. Тестирование демонстрирует наличие дефектов
2. Исчерпывающее тестирование недостижимо
3. Раннее тестирование
4. Скопление дефектов
5. Парадокс пестицида
6. Тестирование зависит от контекста
7. Заблуждение об отсутствии ошибок.

2

# Определения

- Отладка (debugging)
- Требование (requirement)
- Тестовый случай/сценарий (test case/scenario)
- Цель тестирования (test target)

# Тестовый случай

## Input → Processing → Output

- 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)

# Тестовый случай (2)

- Повторяемый, автоматизируемый
- Учитывает состояния (если есть)
  - Переходы между состояниями
    - Правильные: корректный результат
    - Неправильные : корректные сообщения об ошибках
- В российской официальной терминологии используется термин сценарий

# Тестовый 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 <b>верный</b> pin-код	Проверка pin-кода	Приглашение к выбору транзакции	Ожидание выбора транзакции
3	Ожидание выбора транзакции	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

# Тестовые сценарии (2)

- 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 <b>неверный</b> pin-код	Проверка pin-кода	Сообщение об ошибке	Ожидание pin-кода
3	Ожидание pin-кода	Вводим <b>неверный</b> pin-код	Проверка pin-кода	Сообщение об ошибке	Ожидание pin-кода
4	Ожидание pin-кода	Вводим <b>неверный!!!</b> pin-код	Проверка pin-кода, блокировка карточки	Сообщение об ошибке и блокировка	Готов



# Тестовые сценарии (3)

- 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

# Сколько тестов?

- 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

# Анализ эквивалентности

- Набор входных данных на которых результат является эквивалентным или подобным
  - Например, если вывод системы одинаковый — находимся внутри эквивалентной области
  - Дополнительно может использоваться анализ граничных значений
- Банкомат:
  - Купюры по \$100
  - За раз максимум \$2000
  - За день \$10000
  - Комиссия 1% но не меньше \$5



# Анализ эквивалентности (2)

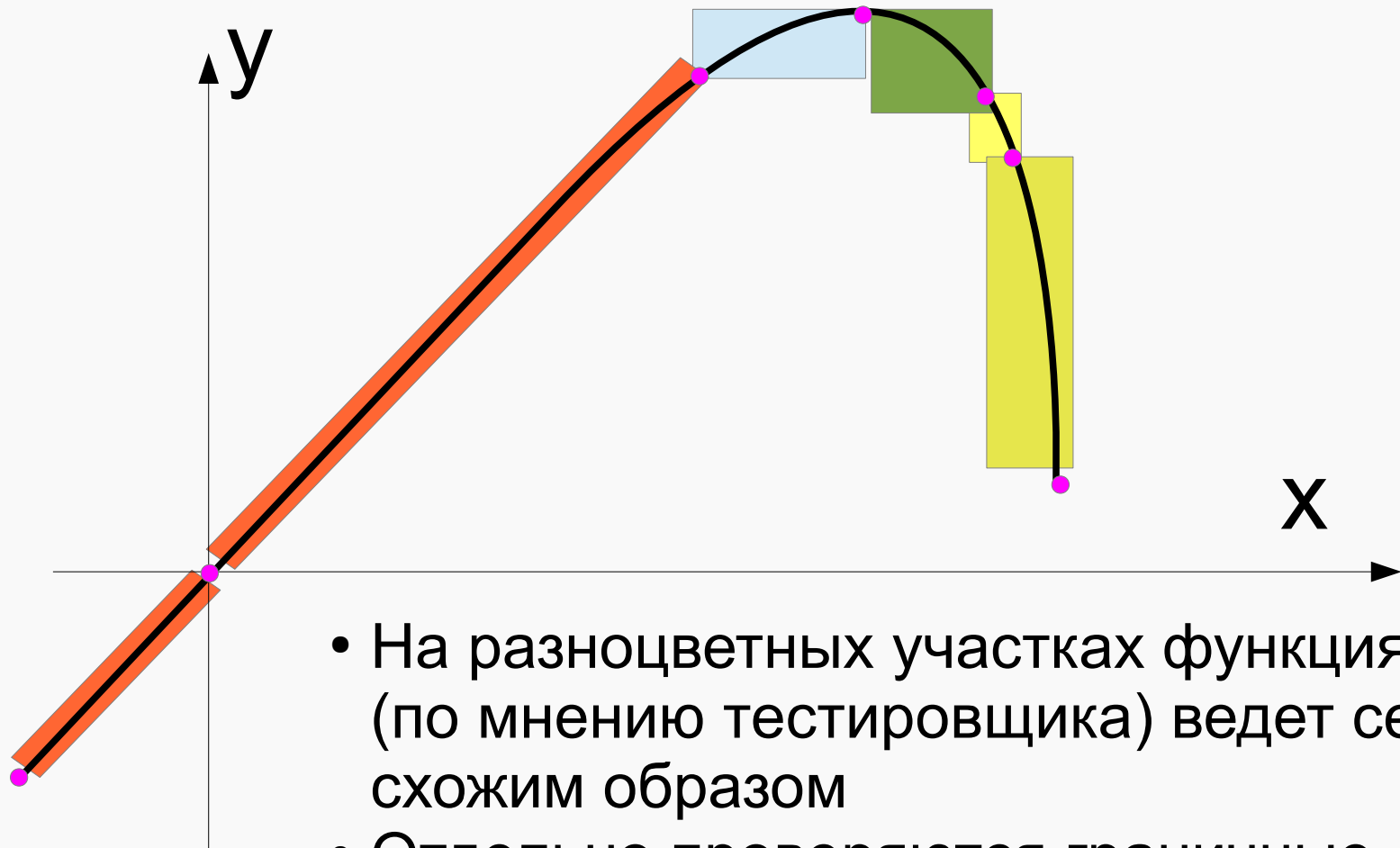
Сумма к снятию	Сумма к списанию	Результат	Класс результата
-1	-	нет	невалидный
1..99,101..199, ...	-	нет	невалидный
100,200,...,500	105,205, ..., 505	выдача	валидный
600,700,...,2000	606,707, ..., 2020	выдача	валидный
2100,2200...	-	нет	невалидный
2000+1..99	-	нет	невалидный
2000+100	2020+105	Выдача в два этапа	валидный
2000+2000+2000+2000+2000	10100	Выдача в 5 этапов	валидный

Граничные значения для параметра выдачи средств

Number Line	-999	99	100	2000	2000	10000
	Invalid		Valid		Invalid	

# Анализ эквивалентности

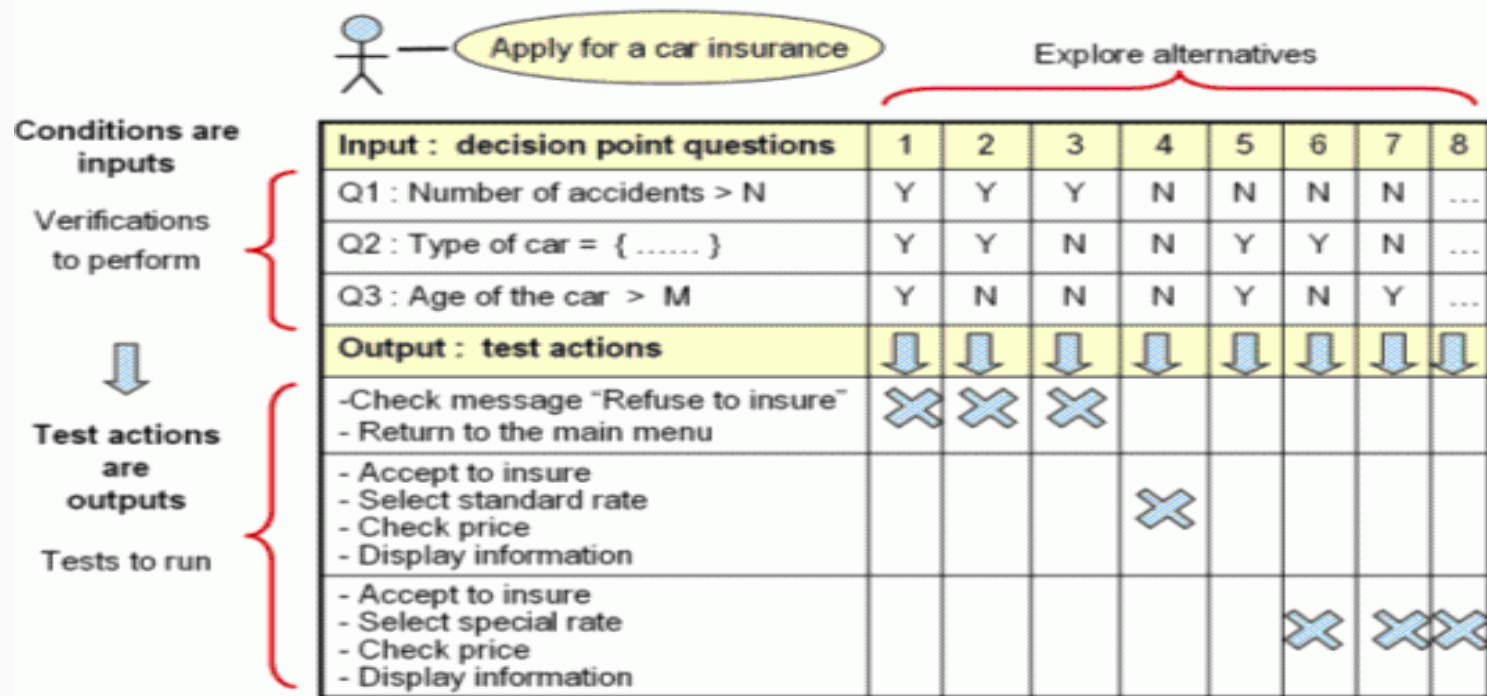
Зависимость удовлетворения от количества съеденного холодца



- На разноцветных участках функция (по мнению тестировщика) ведет себя схожим образом
- Отдельно проверяются граничные значения

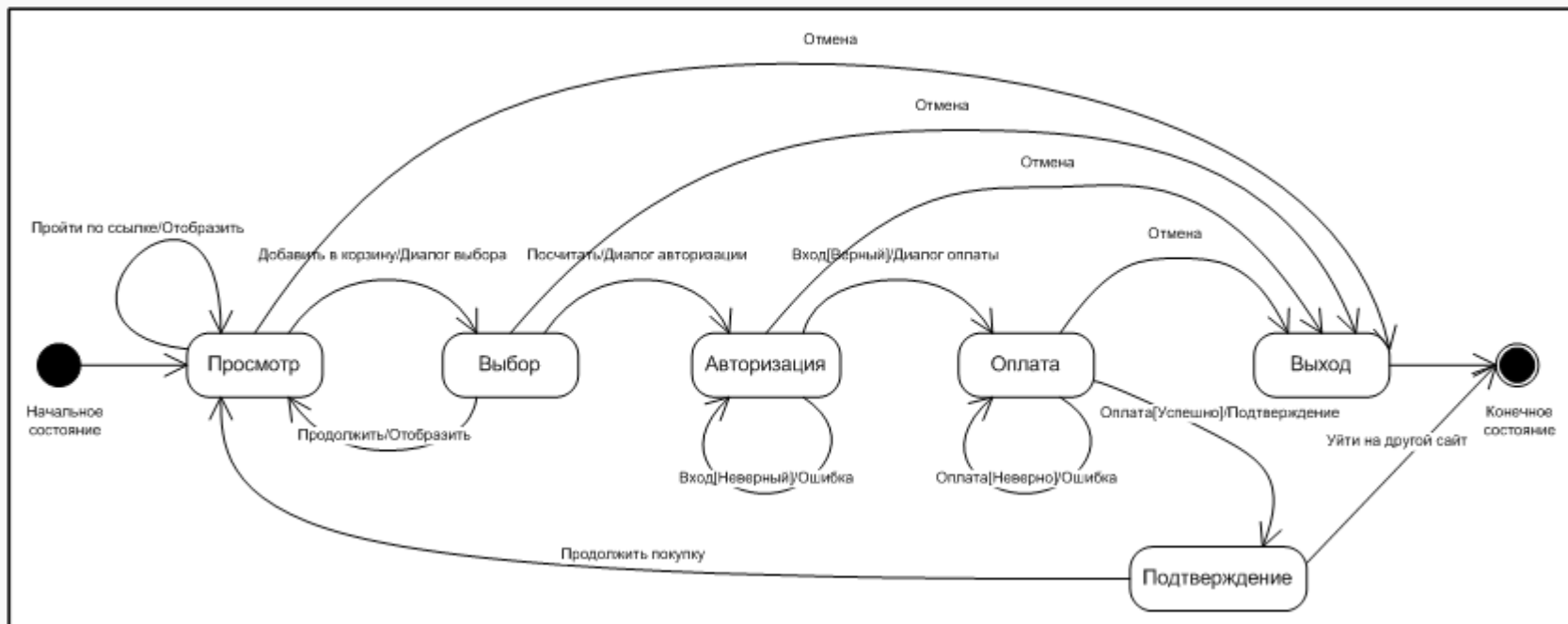
# Таблица решений

- Используется в системах со сложной логикой, описание конечного автомата
- Может включать большой набор условий (обычно true-false), и действий



# Таблицы переходов

- Позволяют выбрать состояния и их комбинации, которые можно опустить
  - Рекс Блэк «Advanced Software Testing».
  - <http://inrecolan.ru/blog/viewpost/361>





# Таблицы переходов (2)

- Прокрыть определенные строки тестами
- while (есть тесты с «непокрытыми» строками)
  - Выбрать состояния «не определено»
  - Попытаться покрыть тестом

Состояния		События/Условия		Текущее состояние	Событие/Условие	Действие	Новое состояние
	X		=	Просмотр	Пройти по ссылке	Отобразить	Просмотр
		Пройти по ссылке		Просмотр	Добавить в корзину	Диалог выбора	Выбор
		Добавить в корзину		Просмотр	Продолжить	Не определено	Не определено
		Продолжить		Просмотр	Выписать	Не определено	Не определено
Просмотр		Выписать		Просмотр	Вход [неверный]	Не определено	Не определено
Выбор		Вход [неверный]		Просмотр	Вход [верный]	Не определено	Не определено
Авторизация		Вход [верный]		Просмотр	Оплата [неверно]	Не определено	Не определено
Оплата		Оплата [неверно]		Просмотр	Оплата [успешно]	Не определено	Не определено
Подтверждение		Оплата [успешно]		Просмотр	Отмена	Нет действия	Выйти
Выйти		Отмена		Просмотр	Продолжить покупку	Не определено	Не определено
		Продолжить покупку		Просмотр	Уйти на другой сайт	Не определено	Не определено
		Уйти на другой сайт		Выбор	Пройти по ссылке	Не определено	Не определено
				{53 строки, сгенерированных по шаблону, не показаны}			
				Выйти	Уйти на другой сайт	Не определено	Не определено

# Сценарии использования (функциональное тестирование)

Прецедент: ViewInformationAboutTheRoute

**ID:** 2

**Краткое описание:** Пользователь просматривает информацию о маршруте

**Главные актёры:** Клиент (или любой другой пользователь)

**Второстепенные актёры:** нет

**Предусловия:**

Пользователю выведен список маршрутов.

Пользователь может быть не авторизован в системе

**Основной поток:**

Прецедент начинается когда Пользователь выбирает конкретный маршрут

Система отображает подробную информацию по выбранному маршруту

- Выбираем сценарий
- Проверяем его

# Автоматизация тестов

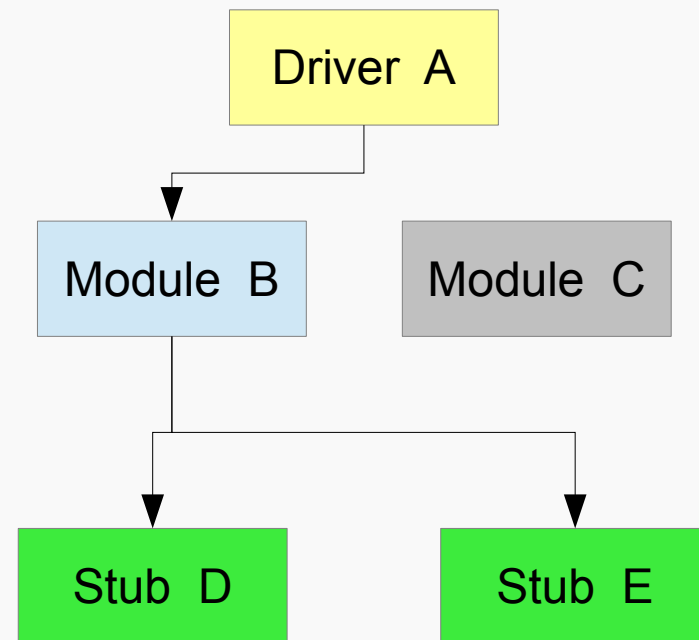
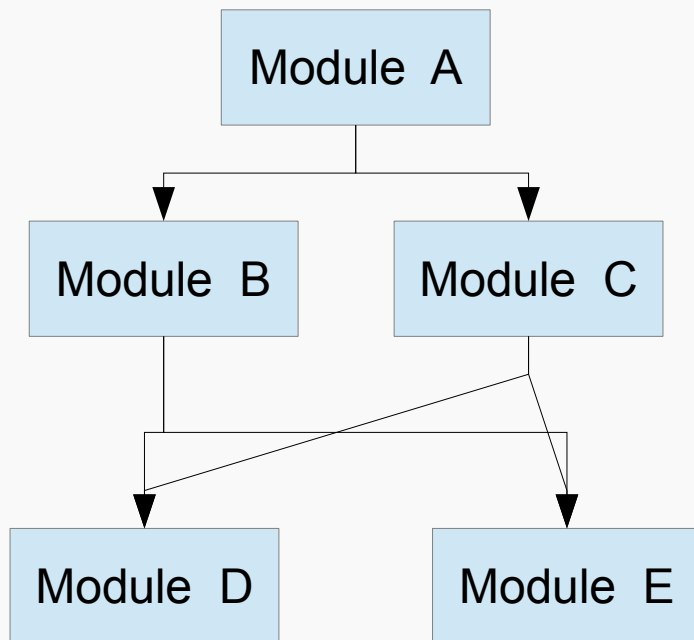
- Регрессионное тестирование
- Повторение тестового сценария
- Приемочное тестирование
- Сокращение ручного труда?
- Проверка одного приложения в разных окружениях

3

- Модульное (компонентное) тестирование - тестирование отдельных компонентов программного обеспечения [IEEE 610].
  - Метод или класс
  - Программный модуль
- Модули описаны в дизайне
- Необходимо изолировать модуль из системы

# Изолирование модулей

- Драйвер вместо вызывающего модуля
- Заглушка вместо подчиненного модуля



# Заглушка

- Эмулирует поведение подчиненной программы
  - Подпрограмма, функция, процедура
  - Аппаратное прерывание, передача данных
- Интерфейс совпадает, внутренность нет
  - Предопределенные ответы для заданных аргументов, исключения
  - Постоянная генерация прерываний
- Используются вместо реальной программы
  - Компилируется и линкуется

# Заглушка (2)

- Простая
  - Нельзя тестировать заглушку!
- Обычно 1 строка кода
- Возможна дополнительная логика
  - 50 раз возвращать 42, на 51 — `IndexOutOfBoundsException`
- Может читать значения из текстового файла
- Возможна настройка драйвером перед выполнением



# Драйверы

- Эмулирует вызываемый компонент
- Обычно уже более сложная программа
  - Устанавливает окружение
  - Подготавливает входные данные
- Дополнительно может:
  - Запускать серию тестов
  - Настраивать заглушки
  - Формировать журнал результатов

# Test Fail

- Для анализа результата тестов программисту необходимо:
  - Входные и выходные значения
  - Значения измененных переменных
  - Пройденные пути, принятые решения
  - Симптомы сбоя
- Где ошибка в тесте или в ПО?

# Фреймворки для модульного тестирования

- XXXunit — много разных!
- Фреймворк vs Библиотека



# JUnit 4

- JUnit фреймворк обеспечивает:
  - Аннотации для маркировки метода как теста `@Test`
  - Аннотации для маркировки действий до и после теста `@Before`, `@After`, `@BeforeClass`, `@AfterClass`
  - Методы для проверки (assertion)
  - UI, журнал тестов...

# Пример

```
import org.junit.*;
import static org.junit.Assert.*;
public class MoneyTest {
    private Whale whale;
    @Before public void setUp() {
        whale = new Whale();
        whale.setLocation("Где-то высоко");
    }
    @Test public void testDown() {
        whale.fallDown();
        assertEquals("Кит не упал",
                    "на земле",
                    whale.getLocation());
    }
}
```

```
@Test (timeout=10)
```

```
public void longLoop() {
```

```
    assertEquals(Pi.computeNNumber(1E10) ,3);
```

```
}
```

```
@Test (expected=IllegalArgumentException.class)
```

```
public void testSqrt() {
```

```
    Math.Sqrt(-5);
```

```
}
```

```
@RunWith(value=Parameterized.class) и @Parameters
```

# @Ignore и автобоксинг

```
long sum(long x, long y) { return x + y; }
```

```
@Ignore("Евгений, помогите!")
```

```
@Test
```

```
public void add() {
```

```
    assertEquals(4, program.sum(2, 2));
```

```
}
```

expected: <4> but was: <4>

- В JUnit4 assertEquals не существует для примитивных типов
  - В Тесте 4 is упакуется в Integer, sum возвращает long
- Сообщение об ошибке значит:
  - expected int 4, but got long 4
- Надо исправить 4 to a 4L
  - assertEquals(4L, program.sum(2, 2));



# Заглушки: Mockito

- dummy object — объект, который передается, но его методы никогда не используются
- fake objects — работающий объект с упрощенной реализацией
- stub — частичная реализация объекта или интерфейса, с целью использования его методов
- mock object — простая имплементация, с предопределенными значениями

# Mockito: Простая заглушка

```
import static org.mockito.Mockito.*;
import static org.junit.Assert.*;

@Test
public void test1() {
    // create mock
    MyClass test = Mockito.mock(MyClass.class);

    // define return value for method getUniqueId()
    when(test.getUniqueId()).thenReturn(43);

    // use mock in test....
    assertEquals(test.getUniqueId(), 43);
}
```

# Mockito: несколько значений

```
@Test
public void testMoreThanOneReturnValue() {
    Iterator i= mock(Iterator.class);
    when(i.next())
        .thenReturn("Mockito").thenReturn("rocks");
    String result=i.next()+" "+i.next();
    //assert
    assertEquals("Mockito rocks", result);
}
```

# Mockito: выборка по значению

```
@Test
public void testReturnValueDependentOnMethodParameter() {
    Comparable c= mock(Comparable.class);
    when(c.compareTo("Mockito")).thenReturn(1);
    when(c.compareTo("Eclipse")).thenReturn(2);
    //assert
    assertEquals(1,c.compareTo("Mockito"));
}
```



# Mockito: возврат значений независимо от значения

```
@Test
```

```
public void
```

```
testReturnValueIndependentOnMethodParameter() {
```

```
    Comparable c= mock(Comparable.class);
```

```
    when(c.compareTo(anyInt())).thenReturn(-1);
```

```
    //assert
```

```
    assertEquals(-1 , c.compareTo(9));
```

```
}
```



# Mockito: возврат значений по типу параметра

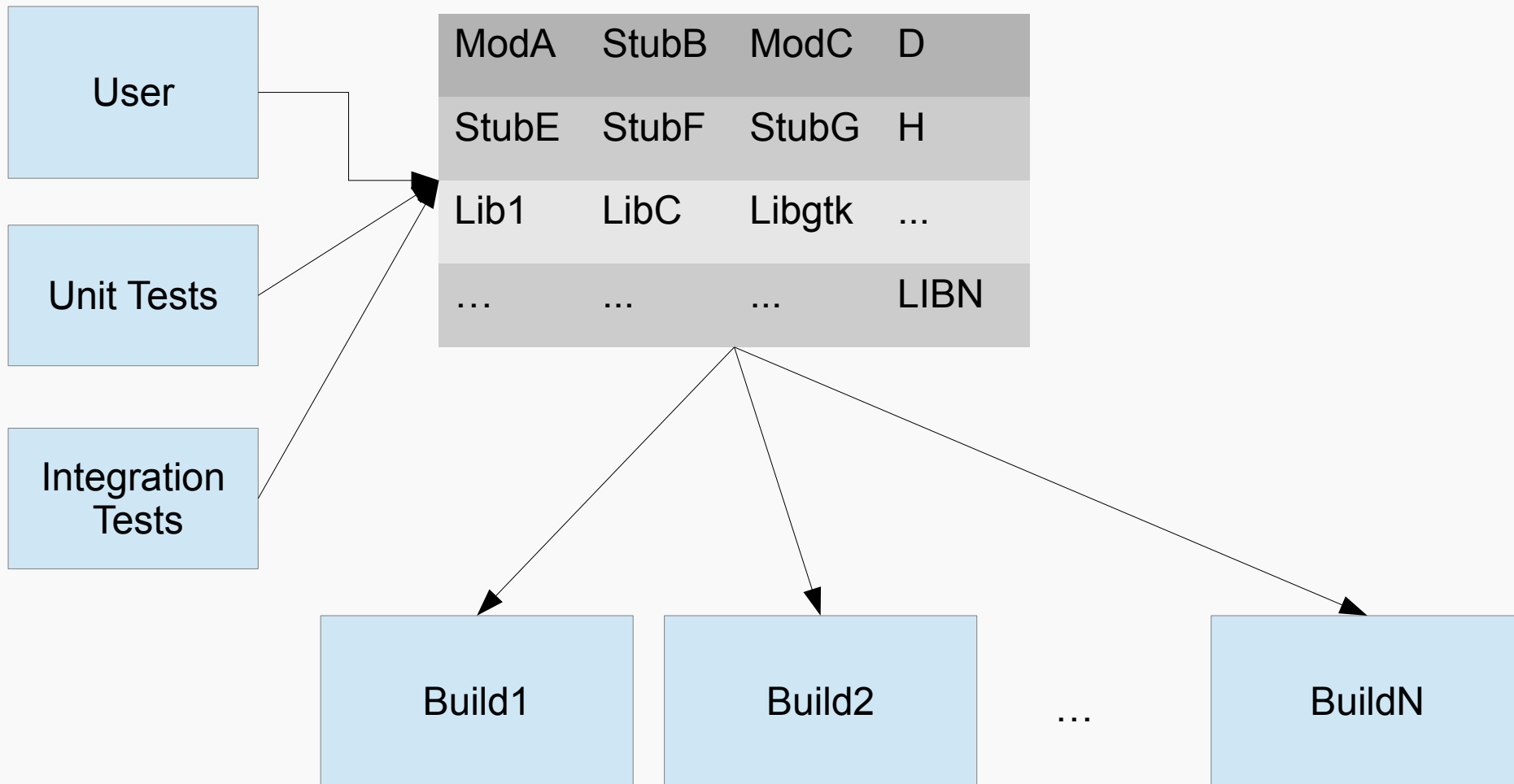
```
@Test
```

```
public void testReturnValueIndependentOnMethodParameter()  
{  
    Comparable c= mock(Comparable.class);  
    when(c.compareTo(isA(Todo.class))).thenReturn(0);  
    //assert  
    Todo todo = new Todo(5);  
    assertEquals(todo ,c.compareTo(new Todo(1)));  
}
```

# Интеграционное тестирование

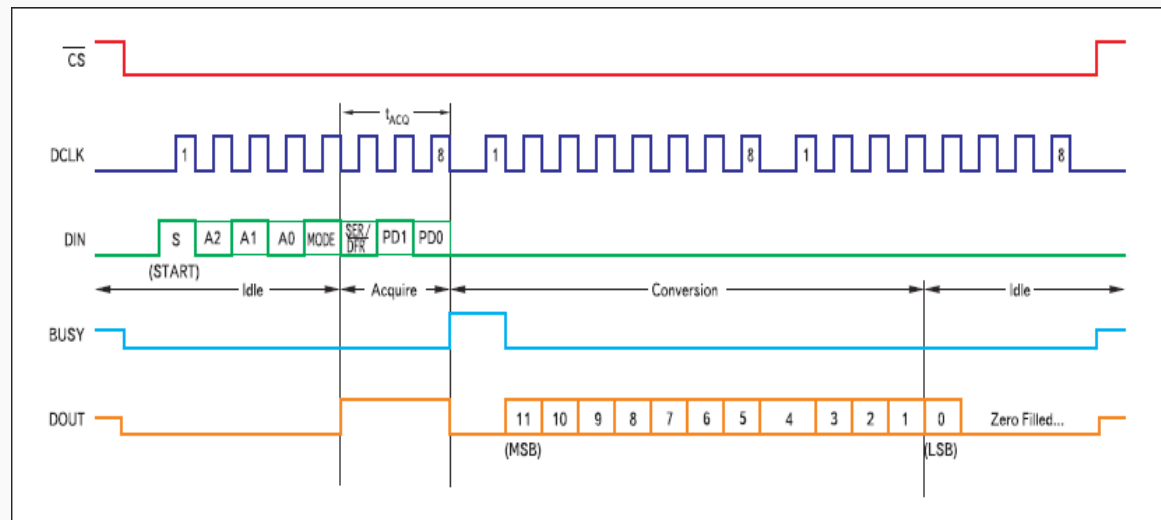
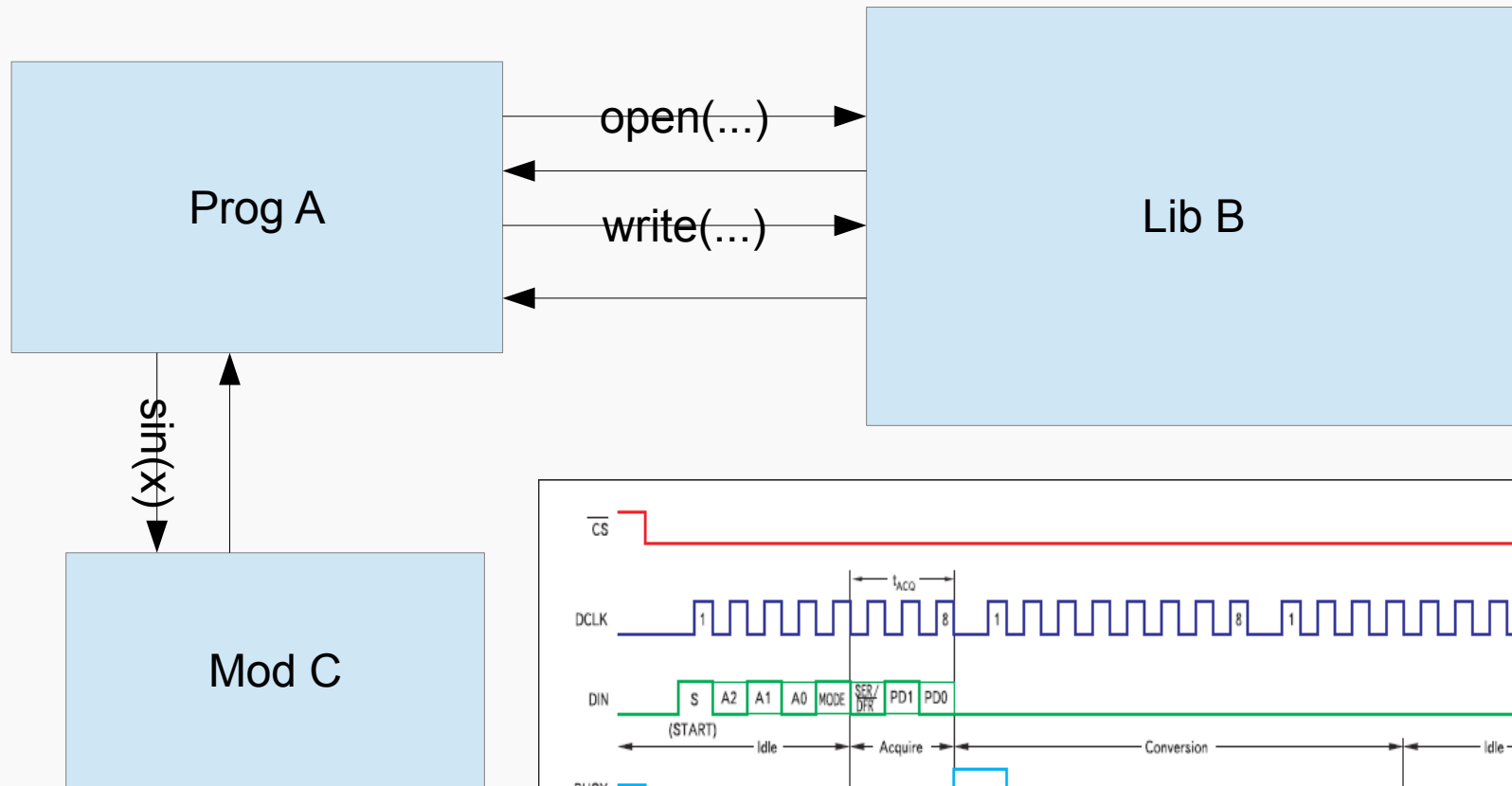
- Проверяет интерфейсы и взаимодействие модулей (компонент) или систем
  - Вызовы API, сообщения между ОО компонентами
  - Баз Данных, пользовательский графический интерфейс
  - Интерфейсы взаимодействия (сетевые, аппаратные, локальные, .... )
  - Инфраструктурные
- Может проводиться когда два компонента разработаны (спроектированы)
  - Остальные добавляются по готовности

# Интеграция

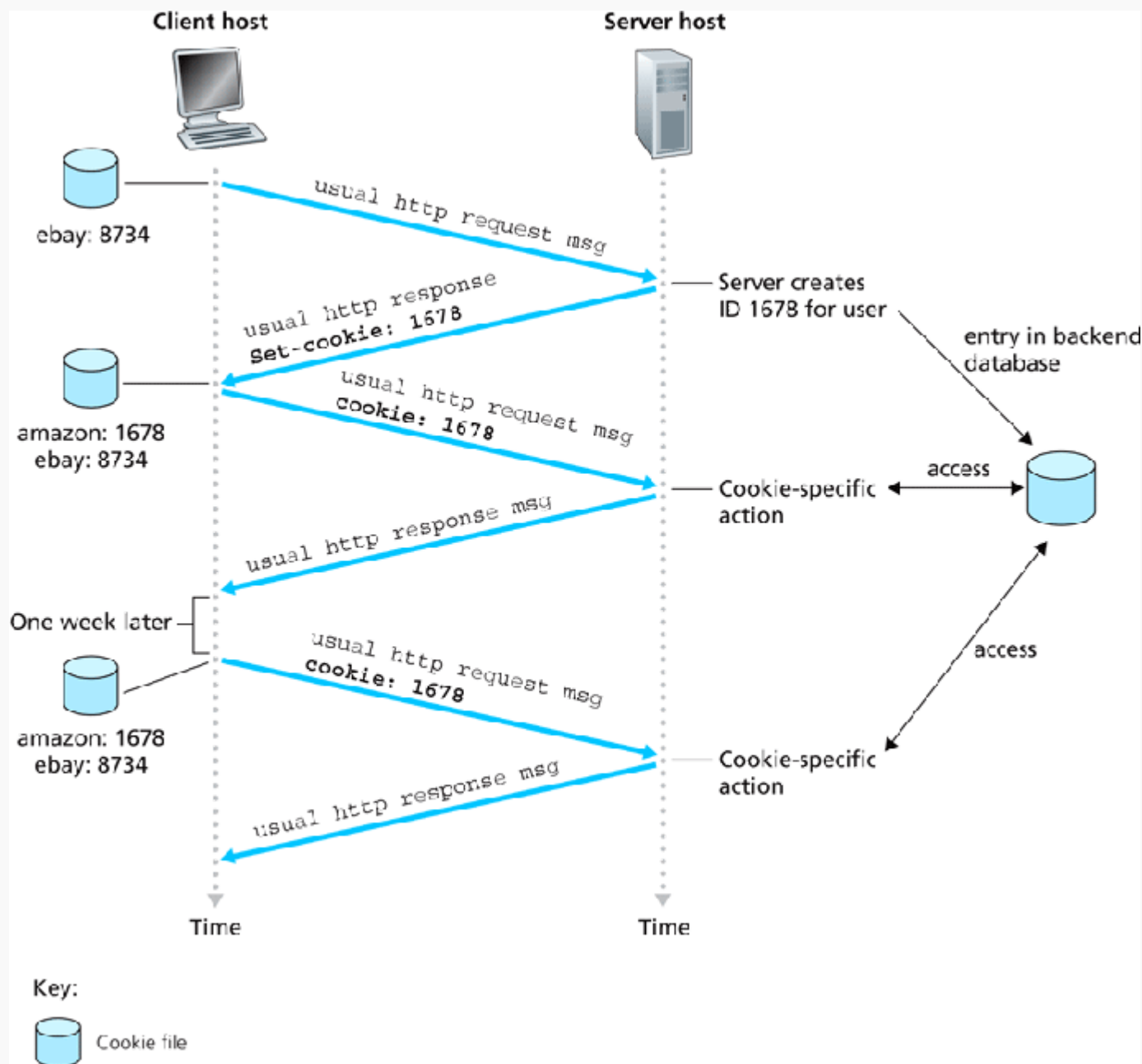




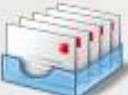
# Вызовы и сообщения



# Сетевое взаимодействие



# Пользовательские интерфейсы



## New Mailbox

☒ Introduction  
☒ User Type  
☒ User Information  
☐ Mailbox Settings  
☐ New Mailbox  
☐ Completion

### User Information

Enter the user name and account information.

Organizational unit:

First name:  Initials:  Last name:

Name:

User logon name (User Principal Name):  
 @e12dom.local

User logon name (pre-Windows 2000):  
 @e12dom.local  
@test.com

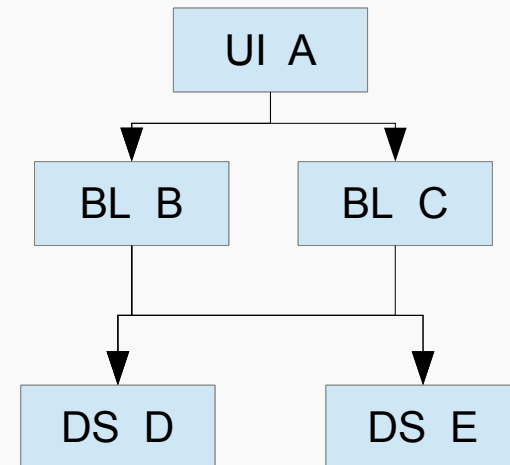
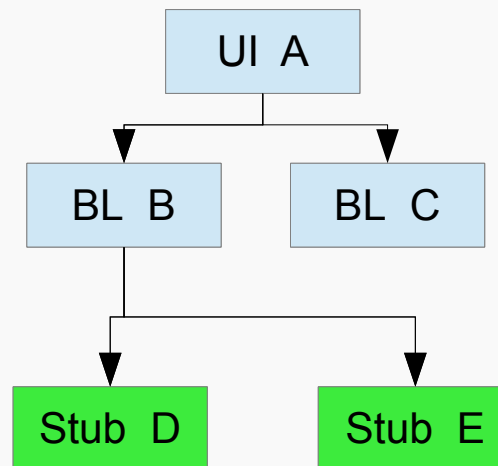
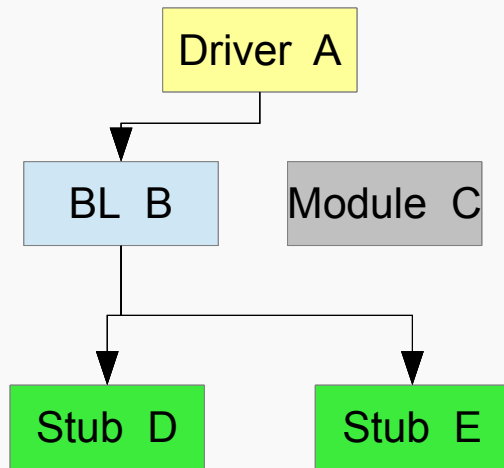
Password:  Confirm password:

☐ User must change password at next logon

# Стратегии интеграции

- Больше объем интеграции — больше сложность
  - Для каждого интерфейса должен быть разработан короткий тест план
- Выбор в зависимости от архитектуры ПО
- Последовательность имеет значение
- Можно тестировать нефункциональные характеристики

# Сверху вниз



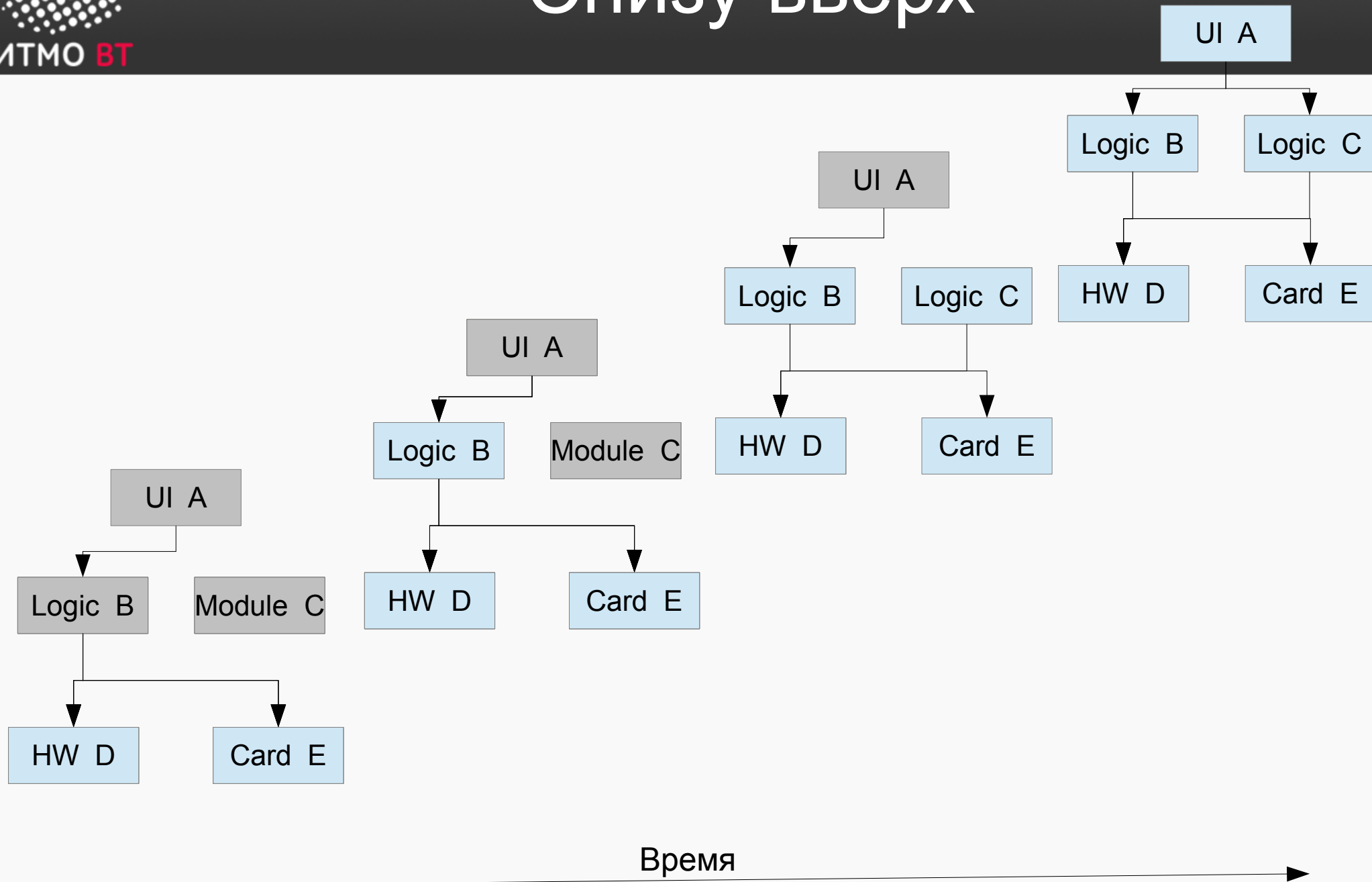
BL — бизнес логика  
 UI — интерфейс пользователя  
 DS — уровень хранения

- + Быстро появляется «осязаемое» приложение
- Большое количество заглушек

Время



# Снизу вверх



- Функциональная (end to end) — по одной функции
  - Собрали 1 сценарий UI-Логика-БД, добавили еще один такой-же
- Ядро (backbone)
  - Экран, клавиатура, мышь работают с минимальными функционалом
  - Добавить цвета на экран, колесо прокрутки ...
- Большой взрыв (big bang)
  - Собрать все вместе и молиться

# Основные принципы

- Интегрировать в первую очередь наиболее сложные компоненты
  - Снизит риски
  - Больше времени для исправления ошибок
- Выбирать порядок интеграции с учетом порядка разработки
- Использовать регрессионное тестирование после каждого этапа интеграции
- Автоматизировать тесты



- На базе сценариев использования
- Ручное/автоматическое
- На готовой системе, в рамках модульного и интеграционного
- Проверяются функции системы начиная с интерфейса пользователя
- Средства автоматизации
  - Открытые: Selenium, Sahi, Watir
  - Коммерческие: от HP, Rational (IBM) ....

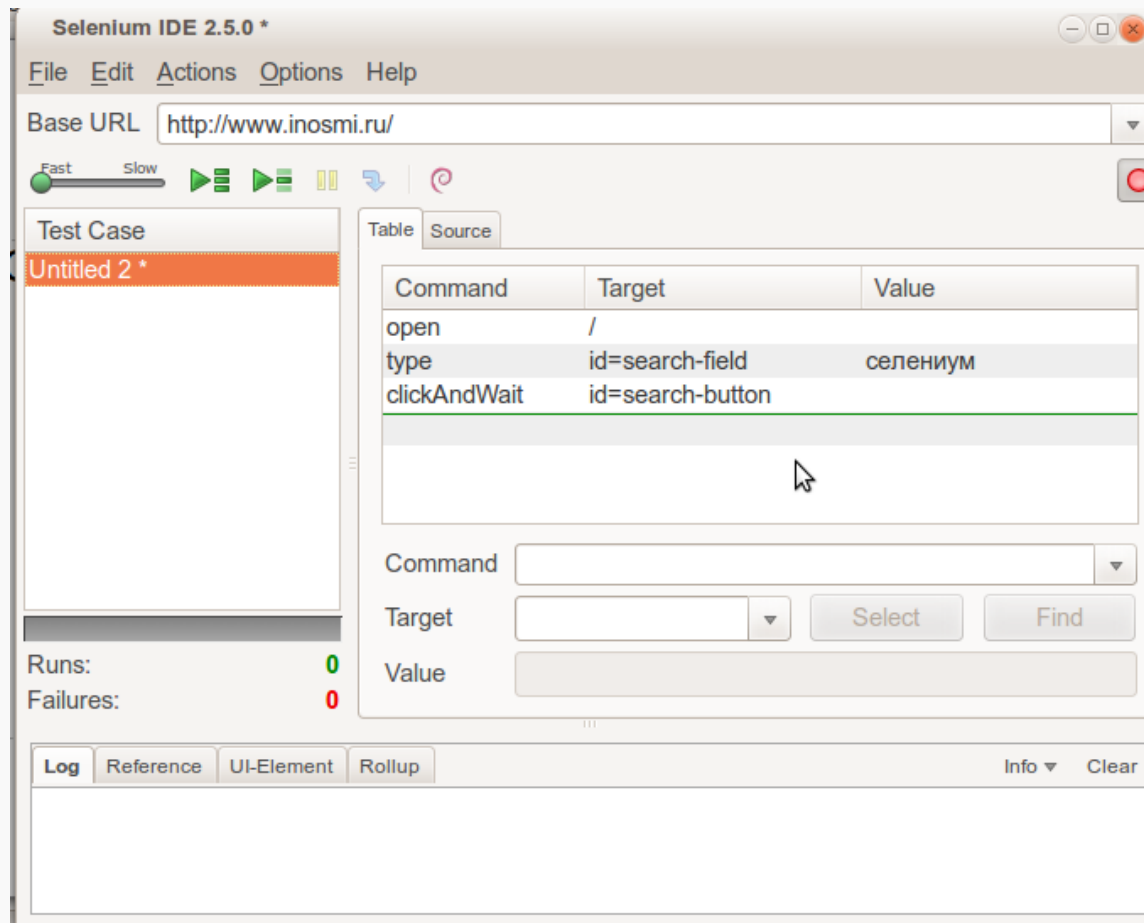
# Selenium

- Набор средств автоматизации
  - IDE
  - Selenium Server / WebDriver
  - Grid
- Тестирование web-приложений
- Кросс-браузерный
- Разработка сценариев на многих языках
  - java, php, python, c#
- Встроенные конструкции assert
- Встроенный механизм логирования ошибок

# Selenium IDE

- Интегрированная среда исполнения и разработки тестов
- Разработано как расширение Firefox
- Запись тестов в виде Java, Ruby, HTML
- Добавление assert и команд

# Selenium IDE



# Команды

- click или clickAndWait — ссылки, переключатели, radio-кнопки
- type — ввод значений
- select — выбор значений из списка
- open — открывает страницу
- assert\*\*\*
- wait\*\*\* - ожидание события
- verify \*\*\* - проверка

# Assertion & Verification

- Предназначены для проверки содержимого элемента UI
  - Элемент присутствует?
  - Есть ли необходимый текст на странице?
- Если verification неуспешна — тест продолжается
- Если неуспешна assertion — тест останавливается

# Добавление

- Добавляется во время записи теста щелчком правой кнопки мыши на элементе
  - `verifyTextPresent`
  - `verifyTitle`
  - `verifyElementPresent`
  - `verifyValue`
- `Assertion` - аналогично

# Синхронизация или WaitFor Command

- `waitForPageLoad(timeout)` загрузка страницы  
ошибка по таймауту
- `waitForAlert`
- `waitForTable` — полная загрузка таблицы
- `waitForTitle` — загрузка заголовка
- Другие команды синхронизации



# Другие команды

- store — сохранение значений в переменной
- echo — запись значения в лог selenium
  - Можно использовать `${var}`

# Запись скрипта в виде кода (фрагмент)

```
public class HelloTest {  
    private WebDriver driver;  
    private String baseUrl;  
    private boolean acceptNextAlert = true;  
    private StringBuffer verificationErrors = new StringBuffer();  
  
    @Before  
    public void setUp() throws Exception {  
        driver = new FirefoxDriver();  
        baseUrl = "http://www.inosmi.ru/";  
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);  
    }  
  
    @Test  
    public void testHello() throws Exception {  
        driver.get(baseUrl + "/");  
        driver.findElement(By.id("search-field")).clear();  
        driver.findElement(By.id("search-field")).sendKeys("селениум");  
        driver.findElement(By.id("search-button")).click();  
    }  
}
```

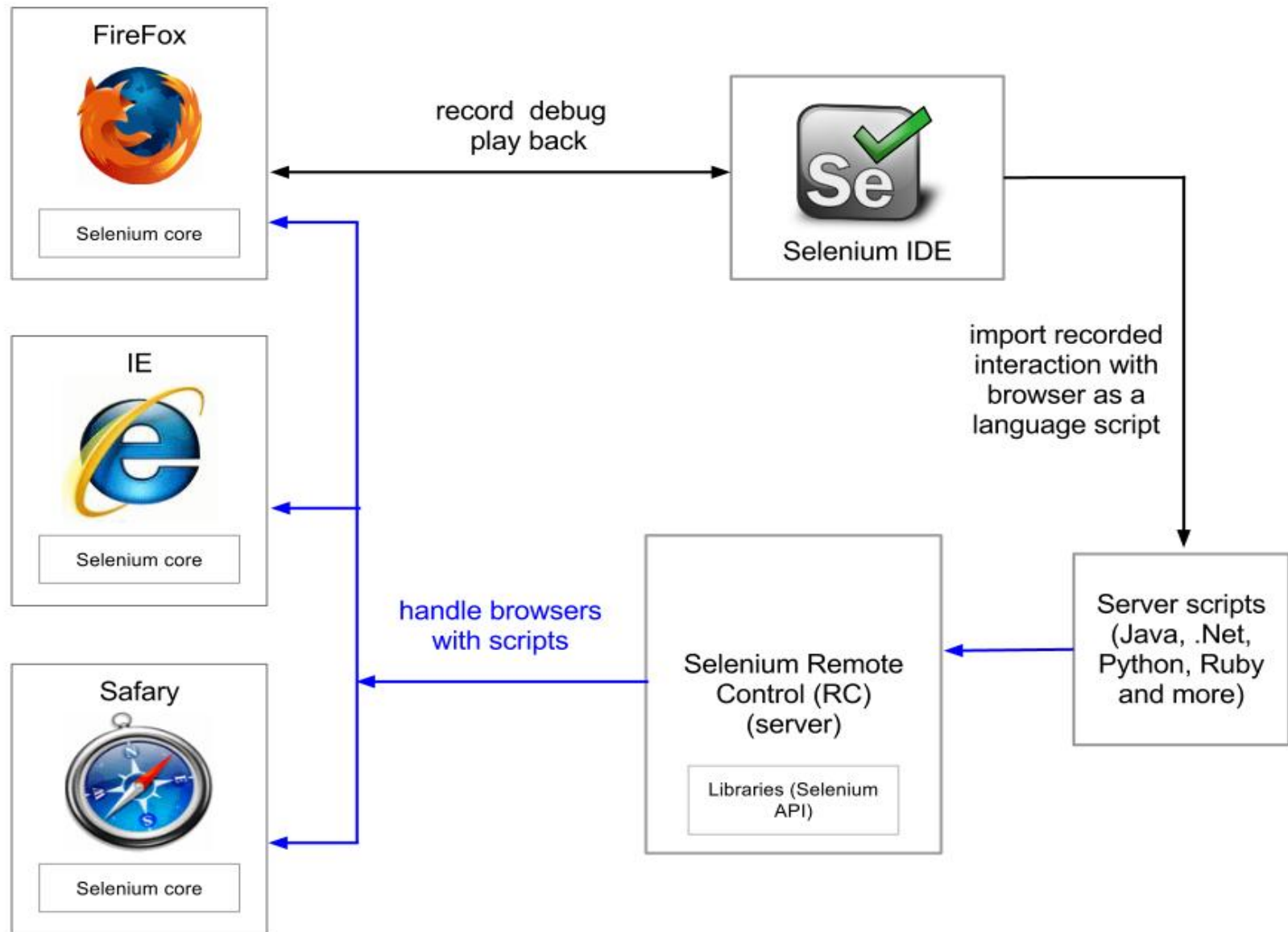
# Ограничения SeleniumIDE

- Запускает тесты только в Firefox
- Слабо развитое управление логикой теста (циклы, условия, ....)
- Запускает только свои сценарии
- Сложно использовать с динамическим содержимым

# Selenium Server

- Может быть использован для кросс-браузерного тестирования
- Сервер для тестирования разработан на java
- Работает как прокси для web-запросов совместно с Webdriver для каждого браузера
- Используется совместно с многими языками программирования
- Разработка — плагины к NetBeans и Eclipse
- Запуск тестов — maven и ant

# Принцип работы

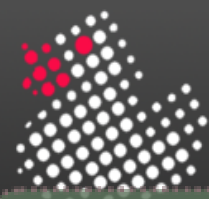


# Процесс разработки

- Записать сценарий в виде java кода
- Создать проект в IDE и добавить необходимые jar файлы (например)
  - junit-4.8.1.jar
  - selenium-java-client-driver.jar
  - selenium-server.jar
  - testng-5.12.jars
- Скопировать сценарий в IDE
- Запустить

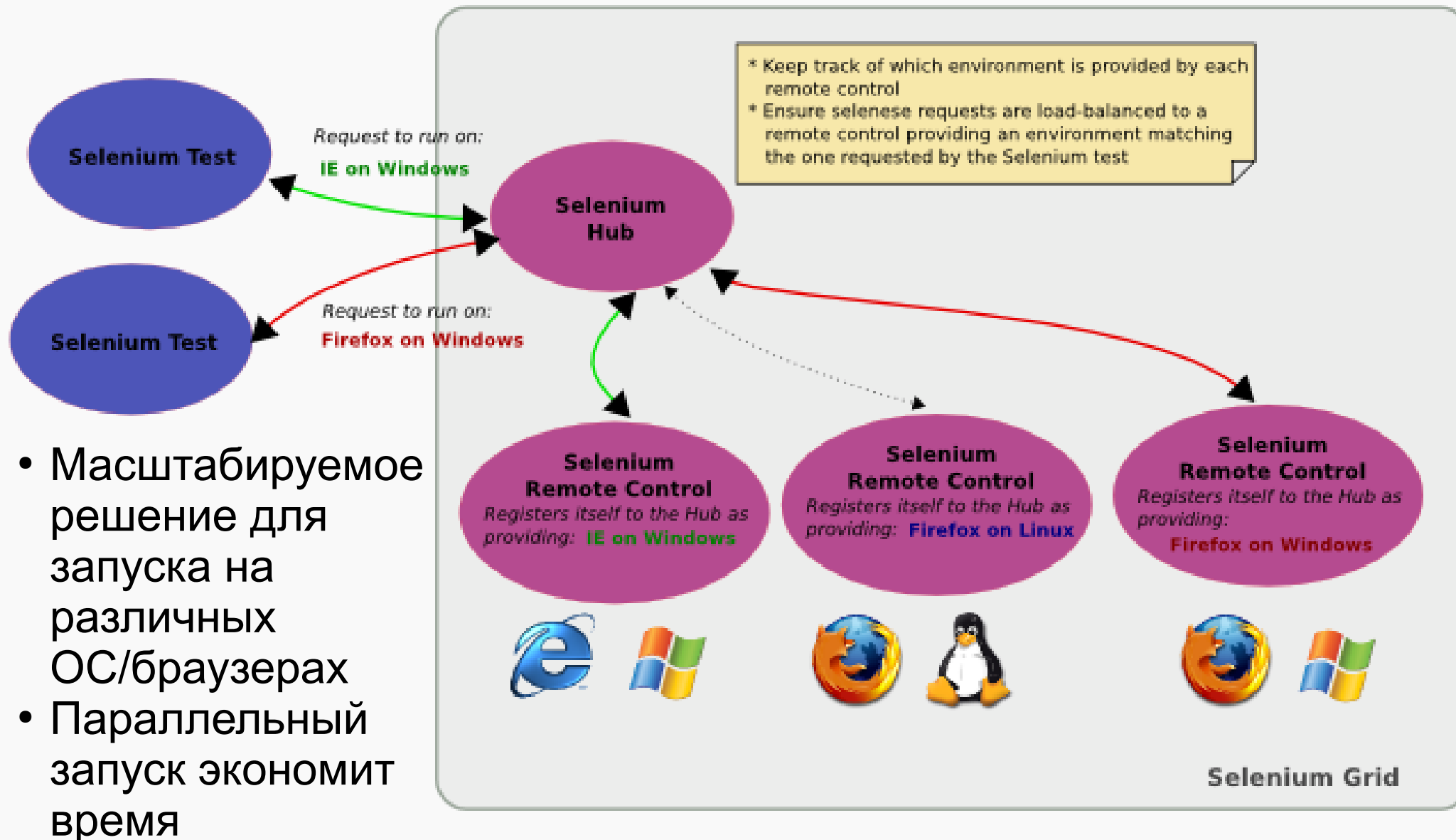
# Xpath локатор

- В современных серверах приложений, порталах нельзя привязываться к компонентам по ID — он динамический
  - Используются классы стилей через xpath
- <http://ru.wikipedia.org/wiki/XPath>
- `//div[contains(@class, 'article-heading')]`



# Selenium Grid

## Selenium Grid : Requesting a Specific Environment





# Автоматизация - не замена ручному тестированию



4

# Статическое тестирование

- Динамические тесты не могут быть исполнены, пока мы не создадим код
- Статические техники (IEEE 1028) могут быть применены до написания кода:
  - «Звонок другу» - неформальные ревью (рецензия)
  - Технические анализ (повторные просмотры)
  - Management review
  - Сквозной контроль
  - Инспекции

# Статическое тестирование

- Может находить ошибки на ранних стадиях разработки!
- Снижение стоимости и рисков
  - Цена ошибки растет с фактором 10 в зависимости от стадии разработки продукта
- Объекты тестирования
  - Политики, стратегии, планы
  - Технические задания, спецификации
  - Артефакты со стадии проектирования, код
  - Планы и подходы к тестированию
  - Прочее...

# Роли в формальном статическом тестировании

- Менеджер (ЛПР)
- Модератор
- Докладчик
- Автор
- Эксперты
- Секретарь

# Неформальные рецензии

- Зачем? - Люди делают ошибки
- Взгляд со стороны — вторые мозги могут помочь найти ошибки
- Цель — найти технические проблемы
  - Не только опечатки!
  - Распространить артефакты, попросить коллег прочитать и комментировать
  - Обычно не очень эффективна — Почему?
  - Формальные методики более эффективны

# Технический анализ

- Проверить продукт на соответствие и практическую пригодность
- Участники анализируют документы предварительно, подготавливают комментарии
- Анализ
  - Предлагает альтернативы и рекомендации
  - Формальный или неформальный

# Сквозной контроль (Walkthrough)

- Проводится автором, который «ведет» аудиторию «через» артефакт
  - Или его части
- Может находить
  - Ошибки, аномалии, неэффективности
  - Спецификации, которые не могут быть проверены
  - Проблемы интерфейсов
  - Отклонения от практик кодирования
  - И пр.
- Может проводиться с образовательными целями



# Инспекции

- 1972, IBM, Michael Fagan — формальный процесс
- Равноправный анализ (systematic peer evaluation)
- Цель — обнаружить и идентифицировать аномалии ПО
- Специально тренированный ведущий (не автор!):
  - Выбирает инспекторов
  - Распространяет заранее подготовленные документы
  - Ведет встречу
  - Убеждается в том, что принятые решения выполнены
  - Определяет и контролирует различные метрики

# Инспекции - 2

- Четко определенные шаги
  - Вход
  - Планирование
  - Обзор
  - Подготовка
  - Обсуждение
  - Переработка
  - Выработка рекомендаций (follow up)
  - Выход
- Повторяется до тех пор, пока все участники включая модератора не удовлетворены

Могут повторяться

# Инспекции — входные и выходные критерии

- Должны быть обязательно определены
  - Не тратить время попусту неподготовленной встречей
- Ведущий убеждается в этом перед началом
- Входные критерии могут быть такими:
  - Подготовлены чеклисты
  - Основополагающие документы вышли из инспекций с известным уровнем ошибок
  - За 10 минут пробной проверки был найден не более одного найденного дефекта
  - Документы грамматически проверены и пр...
- Выходные критерии — документы согласованы

# Планирование инспекции

- Зависит от инспектируемых артефактов
  - Учитывать размер, сложность. Определяется «Check-rate»
  - Разбиение на одновременно обрабатываемые «куски»
- Зависит от участников
  - Убедиться, что все будут присутствовать
  - Обладают ли определенными знаниями и опытом
- Собираемые метрики (н-р: размеры, кол-во найденных дефектов, цена изменений, время на проверку)
- Разрабатывается расписание

# Инспекция: обзор

- Включает:
  - Групповое обучение инспекторов
  - Назначение ролей инспекторам
  - Распространение материалов
- Ведущий представляет инспекторов

# Инспекция: подготовка

- Сердце инспекции — здесь проводится само тестирование
- Инспекторы проверяют артефакты в соответствии с ролью
  - Отмечают время начала
  - Отмечают и записывают проблемы, фокусируясь на основных
  - Используют точно отведенное время (не больше, не меньше — check rate)
  - Классифицируют и считают проблемы (Major, minor, ?)

# Инспекторы - роли

- Человек может фокусироваться только на двух проблемах одновременно
- Роли могут быть выбраны из следующих:
  - Чеклист — инспектор проверят по нему
  - Документы — инспектор проверяет целостность между несколькими документами
  - Фокус — поиск выделенных проблем
  - Перспектива — представить роль и будущее пользователя
  - Процедура — инспектор следует особой процедуре (.)
  - Сценарий — следование заданному сценарию (более специфично, чем предыдущий)
  - Стандарт — проверка на соответствие стандартам
  - Точка зрения — инспектирует с т.з. н-р пользователя



# Инспекция - встреча, обсуждение

- Основное назначение — протоколирование результатов подготовки
  - Позволяет найти еще 10-20% проблем
  - Проблема: id, (Mmon), описание, документ, страница
- Ведущий:
  - Дипломатичен, краток, категоричен
  - Не позволяет развивать дискуссию, длительно предлагать решения
  - Принимает непопулярные решения (удалить к-л, остановить встречу)
- Ограничена по времени — не более 2-х часов
- Решение по артефакту — «принят», «на переработку», «отклонен»



# Инспекция — переработка и выработка рекомендаций

- Если артефакт не принят - отправляется на переработку
  - Автором
  - Редактором
  - Другими работниками
- Инспекция не может завершиться, пока не удовлетворены выходные критерии
  - Проверяется ведущим
- Готовый артефакт передается в систему контроля версий для остальной команды

# Статические методы. Обзор

	Сквозной контроль	Технический Анализ	Инспекция
Основное Предназначение	Поиск дефектов	Поиск дефектов	Поиск дефектов
Дополнительная цель	Обмен опытом	Принятие решений	Улучшение процесса
Подготовка	Обычно нет	Популяризация	Формальная подготовка
Ведущий	Автор	В зависимости от обстоятельств	Подготовленный модератор
Рекомендованный размер группы	2-7	>3	3-6
Формальная процедура	Обычно нет	Иногда	Всегда
Объем материалов	небольшой	От среднего до большого	небольшой
Сбор метрик	Обычно нет	Иногда	Всегда
Выходные данные	Неформальный отчет	Формальный отчет	Список дефектов, результаты метрик, формальный отчет

# Средства статического анализа кода

- Большое количество (C - Lint, Java — FindBugs)
- Находят:
  - Неопределенное поведение (переменная не инициализирована)
  - Нарушение алгоритмов использования библиотеки (fopen без fclose)
  - Сценарии некорректного поведения
  - Переполнение буфера
  - Разрушение кроссплатформенности
  - Дефекты копи-пейста
  - ...

5

# Тестирование системы в целом

- Начинается после окончания интеграции
- Включает несколько фаз:
  - Системное тестирование — выполняется внутри организации-разработчика
  - Альфа- и Бета-тестирование — выполняется пользователем под контролем разработчика
  - Приемочное тестирование — выполняется пользователем.
    - Результат — платить или не платить.
- Методики практически одинаковые, различная строгость интерпретации результатов.

# Системное тестирование

- Стандартная цель
- Фокус:
  - Функциональность системы с точки зрения пользователя
  - Нефункциональные характеристики
- Проводится валидация!
  - В основном метод черного ящика

# Системное тестирование. Напоминание!

- Тестирование — это как научный эксперимент — необходимо повторение
- Автоматизация, скриптинг
- Результаты должны быть:
  - Записаны в журнал.
  - На каждый найденный дефект — зарегистрирован инцидент
  - Периодические отчеты для найденных ошибок

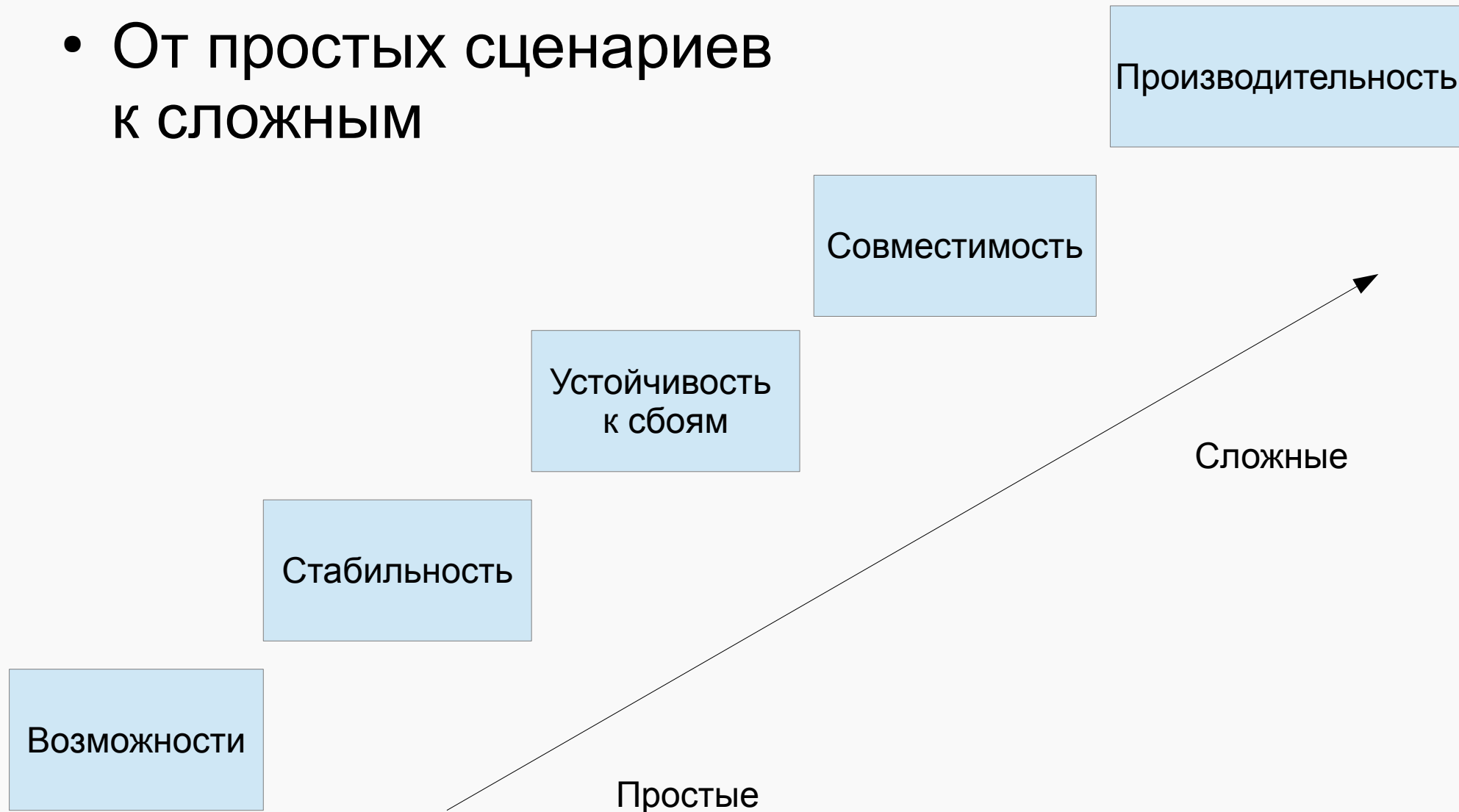
# Контекст системного тестирования

- Тестовое окружение д.б. максимально близко к операционному
  - Оборудование
  - Операционная система
  - Сетевое окружение
  - В аппаратуре — частота, тайминги ... и пр.
- Тестирование эмулирует реальные действия пользователей
  - Тестеры исполняют роли реальных пользователей
  - Скрипты исполняют реальные сценарии
- Система под управлением разработчиков



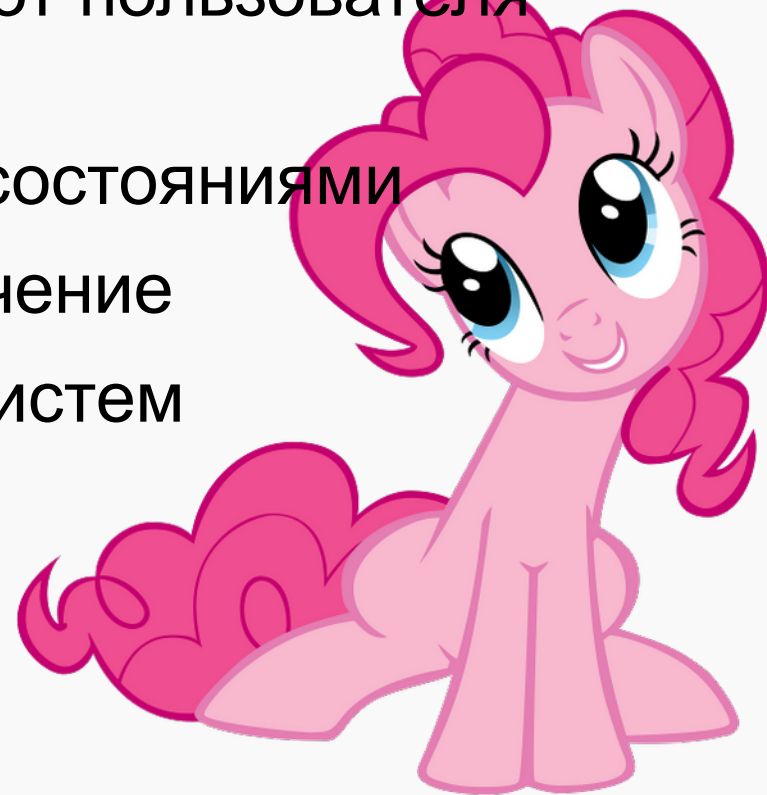
# Где начать?

- От простых сценариев  
к сложным



# Возможности

- Работают ли основные функции системы
  - Один пользователь
  - Одна транзакция
  - Корректный ввод информации от пользователя или из файлов, БД
  - Нормальные переходы между состояниями
  - «Среднее» аппаратное обеспечение
  - Положительные проверки подсистем безопасности



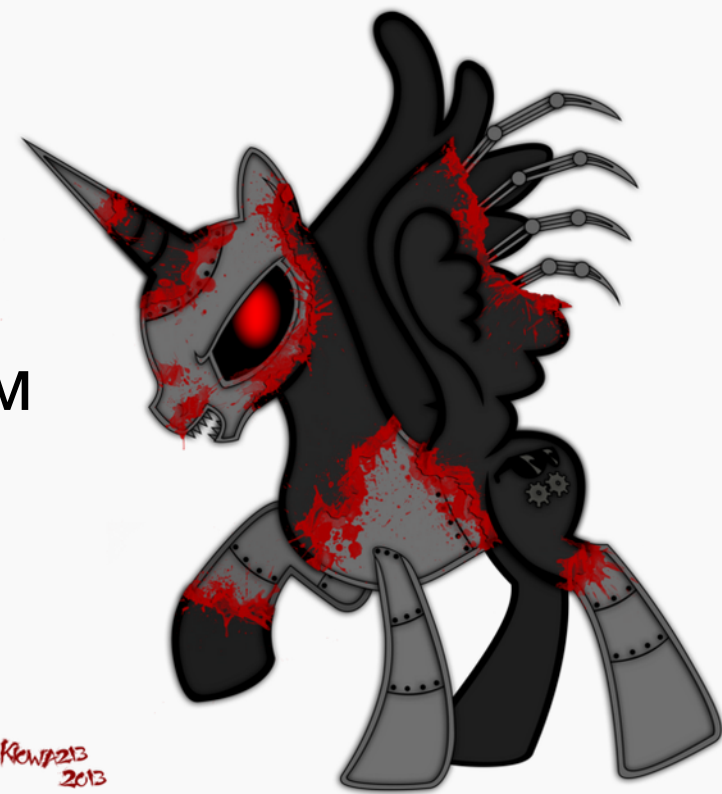
# Стабильность

- Попробуем более реалистичную ситуацию
  - Все еще валидные входные данные и транзакции
  - Несколько пользователей одновременно
  - Реальные последовательности транзакций
  - Одновременно несколько транзакций от одного пользователя (если есть)
  - Запуск системы на длительный период
    - Утечки памяти, переполнения диска. ....



# Устойчивость к сбоям

- Let's break it!
  - Пользователи вводят неправильные данные
  - Сбой сети, испорченные файлы
  - Неправильные переходы между состояниями
  - Выполнение операций в неправильном порядке
  - Аварийное отключение систем/подсистем
  - Намеренные ошибки подсистем безопасности
    - Плохие пароли, внешний взлом



# Совместимость

- Проверяется функционирование в разных средах
  - Взаимодействие с локальными и удаленными системами
  - Проверка функционирования с различными версиями
    - Библиотек
    - Браузеров
    - Операционных систем



# Производительность - CARAT

- CARAT
  - Capacity — Нефункциональные возможности
  - Accuracy — Точность
  - Responce Time — Время ответа
  - Availability — Готовность
  - Throughput — Пропускная способность



- Максимальное количество (пользователей, записей в БД, файлов, Кб, ГГц), поддерживаемое системой
- Одновременно, не нарушая других требований производительности

- Корректность:
  - Алгоритмов
  - Результатов
- Может быть критичной для определенного класса систем

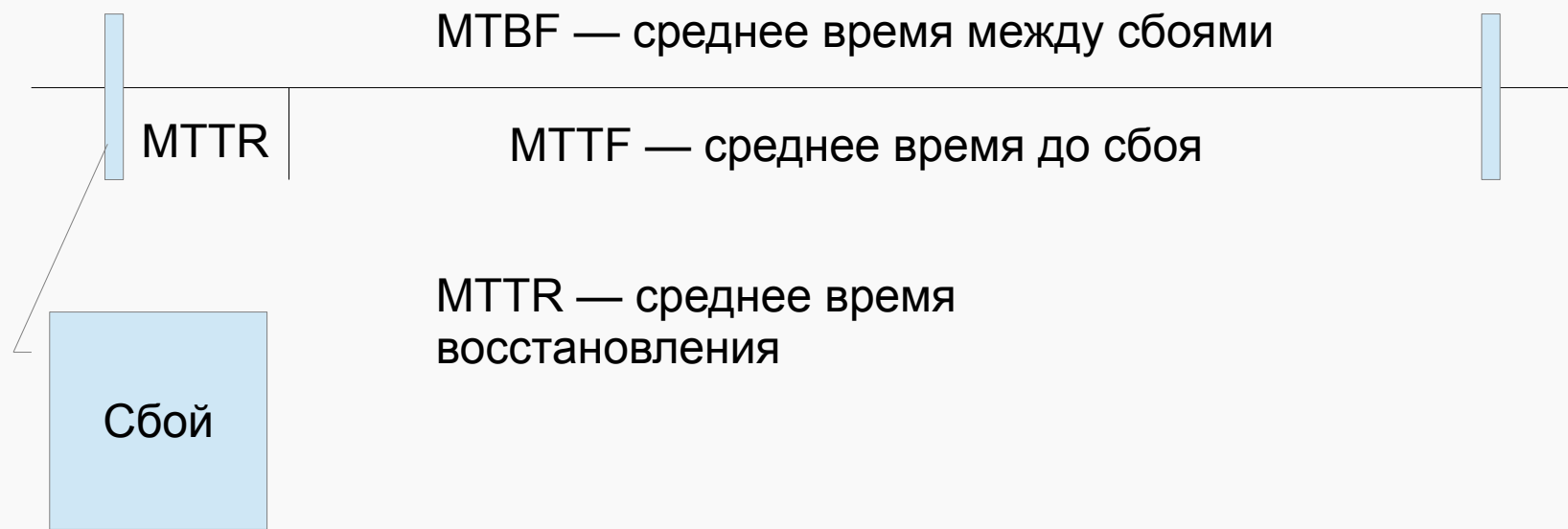


# Время отклика

- Один из наиболее часто измеряемых и критичных критериев
  - Системы реального времени
- От «нажатия на кнопку» до «полной загрузки страницы»
- От подачи управляющего сигнала до наступления реакции
- Под минимальной, расчетной, пиковой нагрузкой

# ГОТОВНОСТЬ

- Коэф. готовности =  $(MTBF - MTTR) / MTBF$



- 0,99999 — сколько минут в год?

# Пропускная способность

- Количество операций (транзакций) в секунду которые может поддерживать система
- Баланс с временем отклика
- Стресс-тестирование — последовательная нагрузка системы до момента неприемлемого времени отклика

- HP Load Runner
- LoadComplete
- IBM Rational Performance Tester
- Load UI Pro
- Apache Jmeter
- The Grinder
- Tsung

# Нагрузочное тестирование с Jmeter

- Бесплатный, кроссплатформенный
- Интерфейс пользователя
  - Простой, строит графики
- Возможность создания распределенной нагрузки
- Эмуляция одновременной работы пользователей
- Снятие метрик
- Возможность разрабатывать плагины
- Планы тестирования в XML — просто контролировать версии

<http://jmeter.apache.org>

# Jmeter и функциональное тестирование

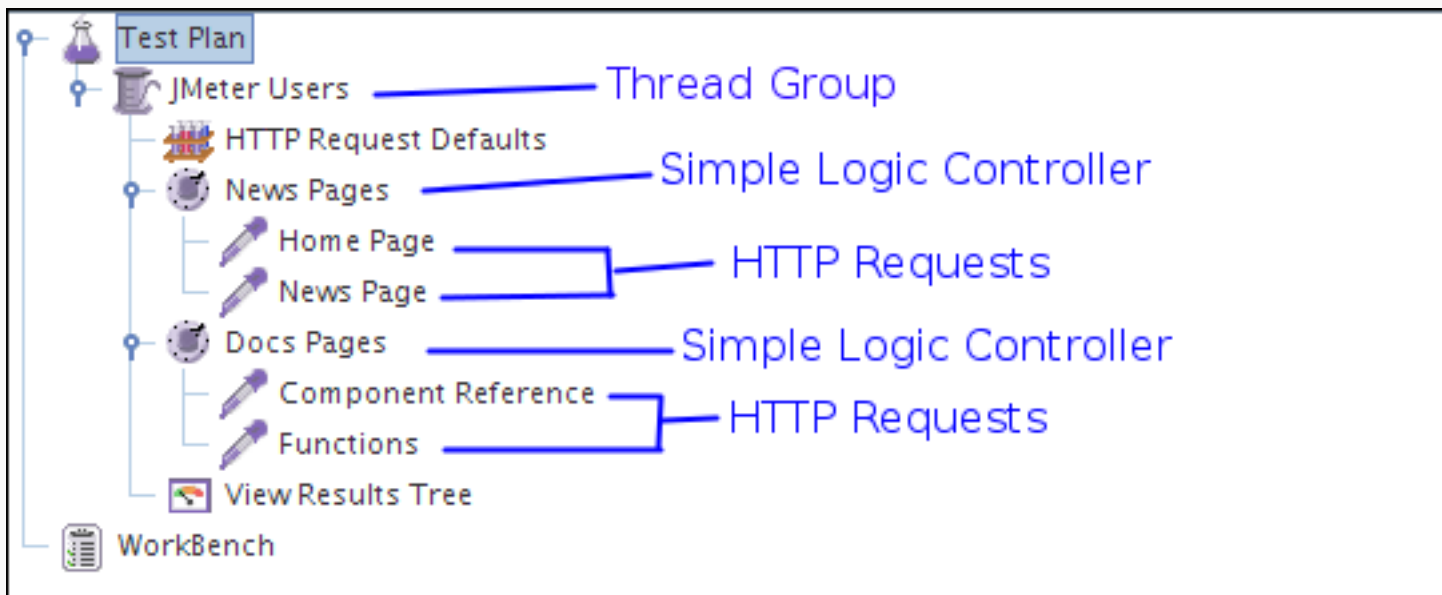
- Может быть использован для записи запросов
- Jmeter → Функциональное тестирование → Нагрузочное тестирование
- Jmeter — не браузер!

# Тестовый план

- Thread Group
  - Описывает пул пользователей для выполнения теста
  - Количество, возрастание и пр..
- Семплы
  - Формируют запросы, генерируют результаты
  - Большой набор встроенных протоколов (TCP, HTTP, FTP, JDBC, SOAP, JMS, SMTP, .....)

# Тестовый план - 2

- Логические контроллеры
  - Определяют порядок вызова семплеров
  - Конструкции управления (if, loop, ...)
  - Управление потоком



[http://jakarta.apache.org/jmeter/usermanual/component\\_reference.html](http://jakarta.apache.org/jmeter/usermanual/component_reference.html)



# Тестовый план - 3

- Слушатели
  - Получают ответы
  - Осуществляют доп. операции с результатами: просмотр, запись, чтение и др.
  - Не обрабатывают данные! (в командной строке, нужен GUI)
- Таймеры
  - Задержки между запросами
  - Постоянные, в соответствии с законами

# Тестовый план - 4

- Assertion
  - Проверяют результаты
- Элементы конфигурации
  - Сохраняют предустановленные значения для семплеров
- Препроцессоры
  - Изменяют семплеры в их контексте (HTML Link Parser)
- Постпроцессоры
  - Применяются ко всем семплерам в одном контексте

[http://jakarta.apache.org/jmeter/usermanual/component\\_reference.html](http://jakarta.apache.org/jmeter/usermanual/component_reference.html)

# Тестовый план — порядок выполнения

- Конфигурационные элементы
- Препроцессоры (Pre-Processors)
- Таймеры (Timers)
- Семплы (Sampler)
- Постпроцессоры (Post-Processors)
- Assertions
- Слушатели (Listeners)

# Jmeter — дополнительные ВОЗМОЖНОСТИ

- Автоматическая генерация CSV-файла
- Bandwidth Throttling - ограничение полосы пропускания
  - Статическое (CPS), динамическое (error rate)
- IP Spoofing — замена `src_ip`, для разделения клиентов
  - Проверка балансировщиков нагрузок
- Поддержка теста TPC-C
  - Стандартный тест на OLTP нагрузку

# Jmeter — распределенное тестирование

## Схема тестирования



# Альфа-тестирование

- Проводится небольшим количеством пользователей внутри организации, разработавшей ПО
  - Может проводиться до окончания системного тестирования
  - Ранние отзывы пользователей об использовании системы в рабочем окружении

# Бета-тестирование

- Проводится выбранной группой пользователей в своем/рабочем окружении
- Windows Beta test — по всему миру
- Могут быть ошибки
- Может быть не реализован весь функционал
- Ранние отзывы для разработчиков
- Превью для пользователей

# Альфа- и бета-тестирование

- Проводится неформально
  - Пользователи работают с тем функционалом, который им интересен
  - ... отмечают то, что не работает
  - ... доводят дефекты и сбои до разработчика
- Нет тестовых сценариев и планов
- Разработчики не гарантируют, что немедленно исправят проблемы. Исправления могут быть:
  - в финальном релизе;
  - в следующей бета-версии.



# Приемочное тестирование

- Ключевой аспект — управляется пользователями
  - А не разработчиками
- Проверка системы на «соответствие предназначению и практическому использованию»
- Формальное
- Неформальное

- Все ли требования удовлетворены
  - Функциональные
  - Нефункциональные
- Источник — документ «требования к системе» (RUP — SRS)
  - Подготовка может начинаться на ранних этапах
  - Может применяться статическое тестирование
- Ответственно подходить к выбору пользователей и обязательно их обучить!

- Окружение должно быть полностью готово
  - Аппаратура и программное окружение
  - Может быть первый запуск в рабочем окружении
- Тесты проводятся и анализируются так же, как и в системном тестировании

6

# Область деятельности

- Фокус - безопасность функциональности, определенной:
  - Спецификациями и требованиями
  - Сценариями использования и моделями
  - Анализом рисков
- Дополнительно внимание уделяется
  - Политики и процедуры безопасности
  - Поведения взломщика/атакующего
  - Известным уязвимостями и дефектам безопасности

# Риски безопасности

- Отдельная группа рисков в разработке и эксплуатации ПО
  - Опасность нарушения конфиденциальности, целостности или доступности информации или информационных систем.
  - Экспозиция риска = функция от вероятности наступления и величины ущерба
  - Стандартные методы работы с рисками: Оценка (assessment), контроль и управление риском.

# Цифровые активы (digital assets)

- Информация и физические объекты
  - Данные пользователей и бизнес-планы
  - Разработанное ПО, документация, модели, диаграммы
  - Документы, процессы, торговые секреты,
  - Налоговая и финансовая информация
  - Презентации и обучающие курсы
  - Сообщения и Emails
  - Информация о работниках
  - Прототипы устройств
  - Возможность оказывать услуги
  - Репутация компании и доверие пользователей

# Методы доступа и обеспечения безопасности

- Типичные методы доступа:
  - Внутрикorporативный LAN и WiFi
  - VPN из публичных сетей
  - Физическая передача объектов (DVD, USB,...)
  - Почта и мессенжеры
- Обеспечение безопасности:
  - Шифрование (криптостойкость, алгоритмы)
  - Аутентификация и токены (сертификаты и политики паролей)
  - Авторизация и права доступа



# Политики безопасности

- Устанавливаются административно.

## Примеры:

- Приемлемое использование, минимальный уровень доступа, управление аккаунтами пользователей
- Классификация информации (публичная, ДСП, секретная, частная, ...)
- Безопасность серверов и мобильных устройств, физический доступ
- Реакция на инциденты, мониторинг безопасности
- Защита от «закладок», тесты безопасности

# Тестирование безопасности

- Как и тестирование в целом, не может найти все дефекты (уязвимости) безопасности
- Зависит от ЖЦ разработки и использования
- Практически используются несколько подходов:
  - Статические анализаторы безопасного кода
  - Фаззинг (Fuzz testing)
  - Тестирование на проникновение (Penetration testing)

# Безопасный код

- Куча стандартов, зависят от языка программирования и базового API
  - SEI CERT Coding Standards (C++, Java,...)
  - OWASP Secure Coding Practices Quick Reference Guide
  - Microsoft secure coding practices
- Статические анализаторы и статическое тестирование позволяют найти основные дефекты
  - OWASP toolset
  - Find Security Bugs (java)
  - ...

# Основные подходы

- CERT Top 10 Secure Coding Practices:
  - Проверяйте все входные значения
  - Следуйте всем предупреждениям компилятора
  - Проектируйте и устанавливайте политики безопасности
  - Keep it simple!
  - Запрещайте доступ если явно не разрешено
  - Реализуйте принцип минимальных доступных привилегий
  - Очищайте данные отправленные к другим системам/модулям
  - Практикуйте многоуровневую защиту
  - Проверяйте качество при помощи тестирования
  - Реализуйте стандарты безопасного кодирования

# Common Weakness Enumeration

- Поддерживается сообществом разработчиков
- Содержит списки типичных уязвимостей с примерами и советами как их избежать
  - CWE Top 25 Most Dangerous Software Errors
  - OWASP Top 10 document
  - CWE Weaknesses in Software Written in C++
  - CWE Weaknesses in Software Written in Java
- Общая база, уязвимости отсортированы по системам, языкам программирования

# CWE-119 Недостаточные ограничения буфера памяти

```
void host_lookup(char *user_supplied_addr) {  
    struct hostent *hp;  
    in_addr_t *addr;  
    char hostname[64];  
    in_addr_t inet_addr(const char *cp);  
  
    /*routine that ensures user_supplied_addr  
    is in the right format for conversion */  
  
    validate_addr_form(user_supplied_addr);  
    addr = inet_addr(user_supplied_addr);  
    hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);  
    strcpy(hostname, hp->h_name);  
}
```

Также может быть: CWE-476: NULL Pointer Dereference

# CWE-79 Cross-site Scripting

- Пример на PHP

```
$username = $_GET['username'];  
echo '<div class="header"> Welcome, ' . $username . '</div>';
```

- Пример на JSP

```
<% String eid = request.getParameter("eid"); %>  
...  
Employee ID: <%= eid %>
```

<https://alf.nu/alert1>

- Что не так?

```
http://trustedSite.example.com/welcome.php?  
username=<Script Language="Javascript">alert("You've been  
attacked!");</Script>
```

# CWE-89 SQL Injection

```
string userName = ctx.getAuthenticatedUserName();  
string query =  
"SELECT * FROM items WHERE owner = '"  
+ userName + "' AND itemname = '" + ItemName.Text + "'";  
sda = new SqlDataAdapter(query, conn);  
DataTable dt = new DataTable();  
sda.Fill(dt);
```

- Можно подставить вместо name:

```
name' OR 'a'='a
```

```
SELECT * FROM items WHERE owner = 'wiley'  
AND itemname = 'name' OR 'a'='a';
```



# CWE — Java (73 уязвимости)

- 5 J2EE Misconfiguration: Data Transmission Without Encryption
- 6 J2EE Misconfiguration: Insufficient Session-ID Length
- 7 J2EE Misconfiguration: Missing Custom Error Page
- 95 'Eval Injection'
- 102 Struts: Duplicate Validation Forms
- 103 Struts: Incomplete validate() Method Definition
- 104 Struts: Form Bean Does Not Extend Validation Class
- 105 Struts: Form Field Without Validator
- 106 Struts: Plug-in Framework not in Use
- 107 Struts: Unused Validation Form
- 108 Struts: Unvalidated Action Form
- 109 Struts: Validator Turned Off
- 110 Struts: Validator Without Form Field
- .....

# Fuzzy testing (Фаззинг)

- Одна из форм анализа уязвимостей
- Запускает программу на множестве аномальных (зачастую случайных) входных данных
- Цель — не дать атакующему воспользоваться эксплойтом в результате сбоя
- Автоматически сгенерировать случайные тестовые случаи
- Мониторинг приложения для поиска ошибок

- Dumb Fuzzing — изменение (mutating) существующих тестов или данных для создание новых тестовых данных
  - Taof, GPF, ProxyFuzz, Peach Fuzzer
- Smart Fuzzing — определение новых тестов на основе моделей входных данных
  - SPIKE, Sulley, Mu-4000, Codenomicon, Peach Fuzzer
- Evolutionary — генерирование тестов на основе ответов от программ
  - Autodafe, EFS, AFL

# Penetration Testing

- В общем случае ничем не отличается от хакинга, что незаконно
- Есть турниры и разборы в сети для гиков (<https://ctftime.org/>)
- Сложно (да и не нужно!) воспитать своего специалиста *по тестированию*
  - Исторически, они «фрилансеры»
- Требуется заключение полноценного договора с группой «тестеров»
- Много типов (auth, ddos, password steal, ...)

# Censored

~~~~~

~~~~~

~~~~~

~~~~~

~~~~~

~~~~~

~~~~~

# Dynamic Application Security Testing (DAST) Tools

- Важны на стадии продуктивного использования для известных движков
  - WordPress, Joomla, Drupal, ...
  - Только известные уязвимости!
- Незаконно не для себя ! Только с письменным разрешением!
- Множество коммерческих продуктов
  - ManageEngine Vulnerability Manager Plus, Paessler PRTG, Rapid7 Nexpose....
- Opensource (OpenVAS, OWASP, WAVSEP, ... есть CE версии коммерческих)

# Организация тестов безопасности в цикле разработки

- Обычный подход, как и для всех тестов
  - Планирование: цель — определение сферы тестирования в соответствии с рисками
  - Анализ и проектирование: определение угроз и рисков на базе аудита требований и известных уязвимостей
  - Реализация и выполнение тестов (с учетом перспективы внутренних и внешних пользователей, взломщиков, и т. д.)
  - Отчеты по результатам
  - Интеграция тестов в процесс разработки.

# Особенности при последовательной разработке

- Раннее определение рисков и требований к безопасности
- Требования безопасности могут меняться, но не отражаться в требованиях
- Обычно выполняются в конце проекта
- Сложность исправления найденных проблем





# Особенности при итеративной или инкрементальной разработке

- Потребности в тестах безопасности возникают во время проекта (н-р во время спринта)
- Возможно изменение (включая полное) подходов
- Тесты могут проводиться в течении всего проекта
- Стратегия на избегание риска может не сработать за один релизный цикл



# Тестирование общих механизмов безопасности

- System Hardening («закалка») — удаление всего лишнего, добавление ПО безопасности
  - Тесты могут проводить аудит системы и прикладного ПО (пароли по умолчанию, лишняя конфигурация, известные дыры в библиотеках...)
- Аутентификация и Авторизация — наиболее часто ломаемый модуль
  - Тестируем брутфорс паролей, фильтры на ввод (XSS, injection), доступность URL
- Шифрование
  - Тесты на дизайн, разработанный код, конфигурацию



# Тестирование общих механизмов безопасности (2)

- Брандмауэры и Зоны
  - Тесты могут сканировать порты, отсылать битые пакеты, dos, ddos, фрагментация
- Средства обнаружения взлома
  - Различное инвазивное изменение данных и сообщений
- Средства обнаружения вредоносного ПО
  - Нет смысла в реальных врусах - “Eicar” (anti-malware test file)
- Обфускация данных
  - Реверс инжиниринг, брутфорс (н-р unXOR)