



Автоматизация тестирования на практике

Про меня

Руководитель группы тестирования,
отдела разработки и внедрения систем
on-line прое



Что такое тестирование



Цель Тестирования



Для чего нужно тестирование?



Тестирование нового функционала

Мы хотим быть уверены, что новый функционал работает именно так, как было нужно заказчику.

Тестирование «старого» функционала

Мы хотим быть уверены, что после внесения изменений остальная функциональность программы не пострадала. Все работает в соответствии с ранее утвержденными требованиями.

Это обычно называют регрессионным тестированием.

Про стандарты

ГОСТ Р ИСО/МЭК 9126-93.

**Информационная технология. Оценка
программной продукции.**

**Характеристики качества и
руководства по их применению**

Классификация видов и направлений тестирования



По запуску кода на исполнение

По запуску кода на исполнение ↗

Статическое

Тестирование требований

Code review

Динамическое

Модульное тестирование

Интеграционное тестирование

Приёмочное тестирование

По доступу к коду и архитектуре приложения

По доступу к коду
и архитектуре приложения ↗

Метод белого ящика

Юнит-тестирование

Полный доступ к коду и архитектуре

Метод чёрного ящика

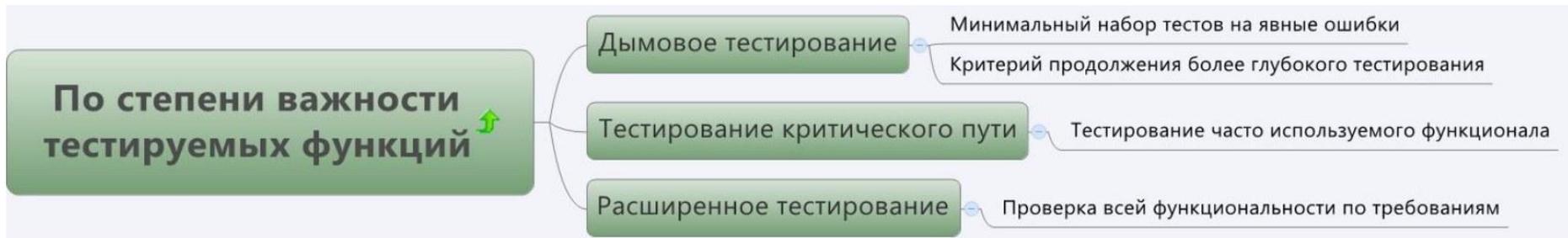
Тестирование через интерфейс пользователя

Тестирование по требованиям

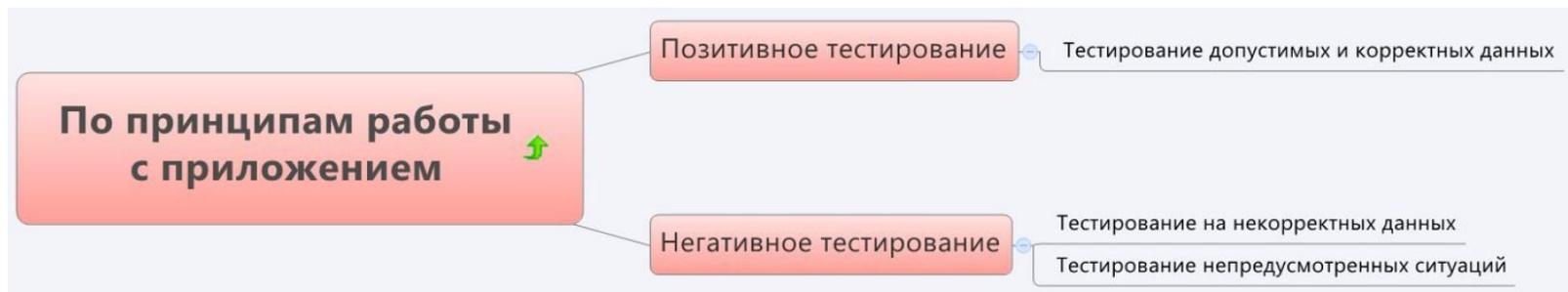
Метод серого ящика

Частичный доступ к структуре приложения (БД)

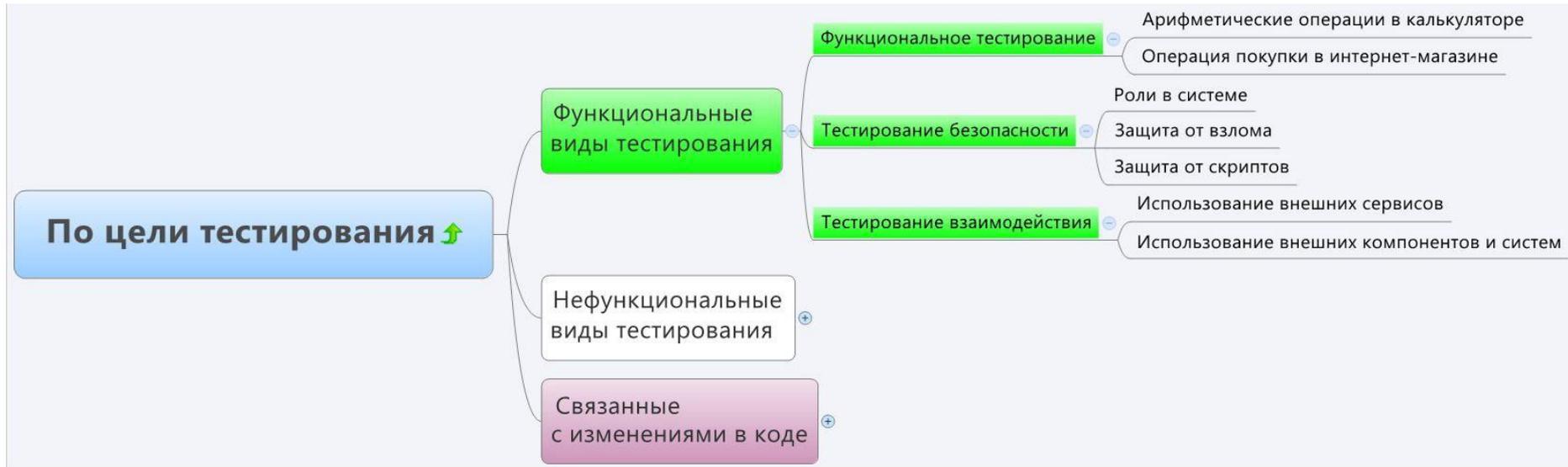
По степени важности тестируемых функций



По принципам работы с приложением



По цели тестирования



Тестирование производительности (Performance testing)

Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой, при этом происходит:

- измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
- определение количества пользователей, одновременно работающих с приложением
- определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций)
- исследование производительности на высоких, предельных, стрессовых нагрузках

По степени автоматизации

По степени автоматизации ↑

Ручное тестирование

Выполнение тестов человеком

Тестирование происходит медленно

Человека нельзя клонировать

Человек сам модернизирует тесты

Автоматизированное тестирование

Тесты выполняет программа

Тесты выполняются быстро

Одни и те же тесты можно выполнять на разных конфигурациях

Автоматизированные тесты нужно поддерживать и модернизировать

Ручное тестирование



Что это

Ручное тестирование-часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения.

Оно производится тестировщиком без использования программных средств, для проверки программы или сайта путём моделирования действий пользователя.

Для чего используется?

- Тестирование нового функционала
- Подготовка сценариев для автоматизированного тестирования

АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ



Что это?

Автоматизированное тестирование программного обеспечения - это процесс верификации программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования.

Для чего используется?

- Регрессионное тестирование
- Тестирование соблюдения контрактов

По уровню детализации приложения (пирамида)

По уровню детализации приложения ↗

Модульное тестирование

Тестирование отдельного модуля ПО

Юнит-тестирование

Интеграционное тестирование

Тестирование связи между модулями

Тестирование связи между частями ПО

Системное тестирование

Тестирование функциональных требований системы

Из чего состоит?



Системное тестирование

Это тестирование программного обеспечения , выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям.

Обычно такое тестирование проходит по утвержденным сценариям, которые описывают последовательности действий для проверки ключевых функций системы.

Плюсы системного тестирования

- Тесты быстро окупаются
- Тесты действуют максимально приближенно к действиям пользователя
- Один тест находит много багов
- Можно подключить к большинству программ
 - Потому что тесты часто представляют собой отдельный проект и не зависят от кода самих систем

Минусы системного тестирования

- Для разработки требуются совместные усилия всей команды (аналитиков, разработчиков, тестировщиков)
- Тесты достаточно дорогие для поддержки/разработки, поэтому обычно тестируется только критичный функционал
- Сложность локализации ошибки
- Протестировать мы можем только то, что система выставляет в виде API или визуального интерфейса

Интеграционное тестирование

Одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе

Тестирование соблюдения контрактов

Именно на этом уровне мы проверяем ключевые контракты с внешними системами и внутри самой системы

На этом уровне мы уже находимся в проекте самой информационной системы, поскольку мы тестируем контракты с внешними системами от её имени под её настройками а также внутренние контракты самой системы

Контроль соблюдения контрактов

После интеграции этих тестов с CI мы получаем возможность оперативно получать информацию о их нарушении.

Для таких тестов создаются отдельные планы, которые запускаются с установленной периодичностью, например раз в 30 минут.

Сообщения будут приходить всем, кто подписался на наблюдение плана в Bamboo

Модульное тестирование

Фаза тестирования, позволяющая проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии

Плюсы модульного тестирования

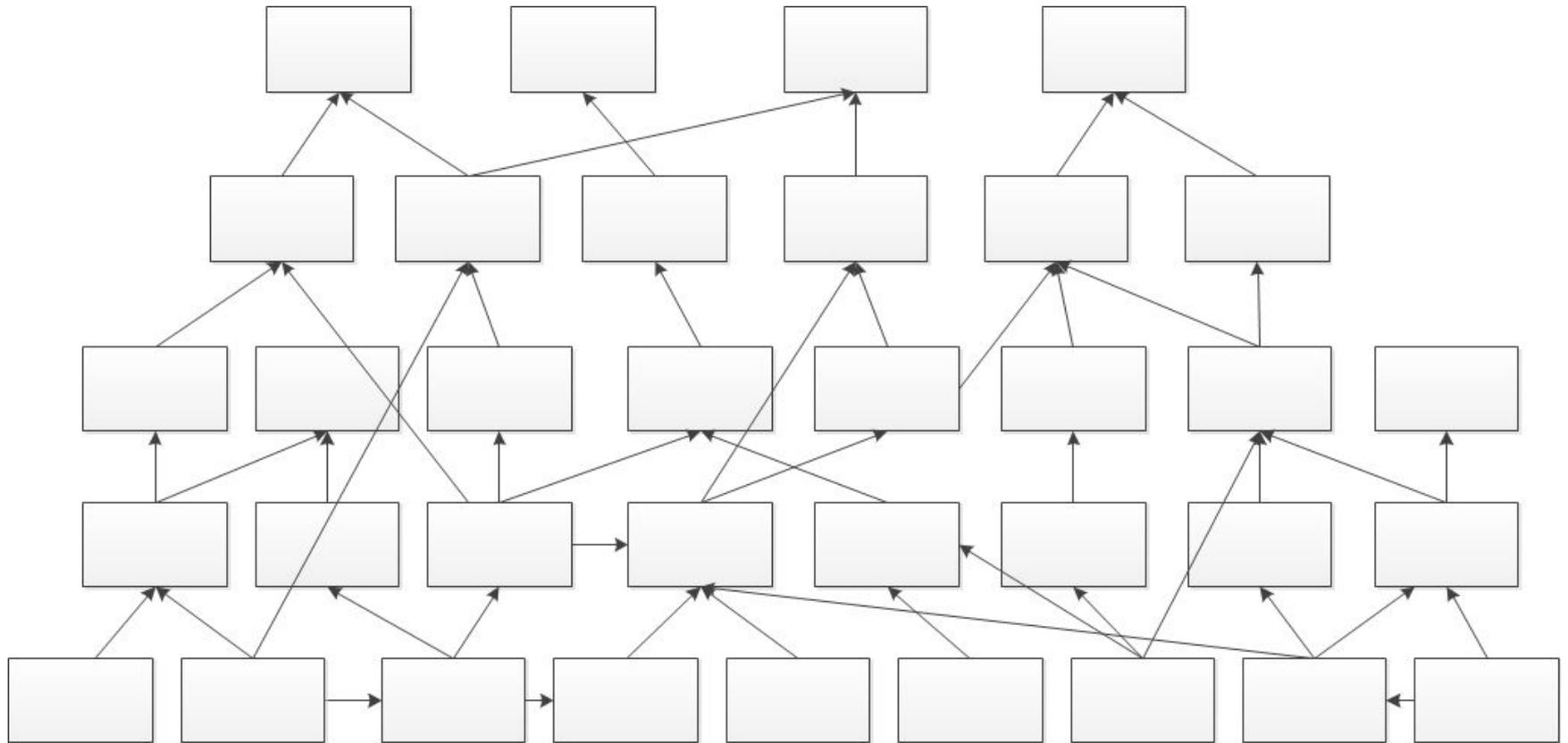
- Легко пишутся
- Высокая локализация ошибки
- Быстро выполняются
- Легко поддерживать
- Повышают качество кода программы

Минусы модульного тестирования

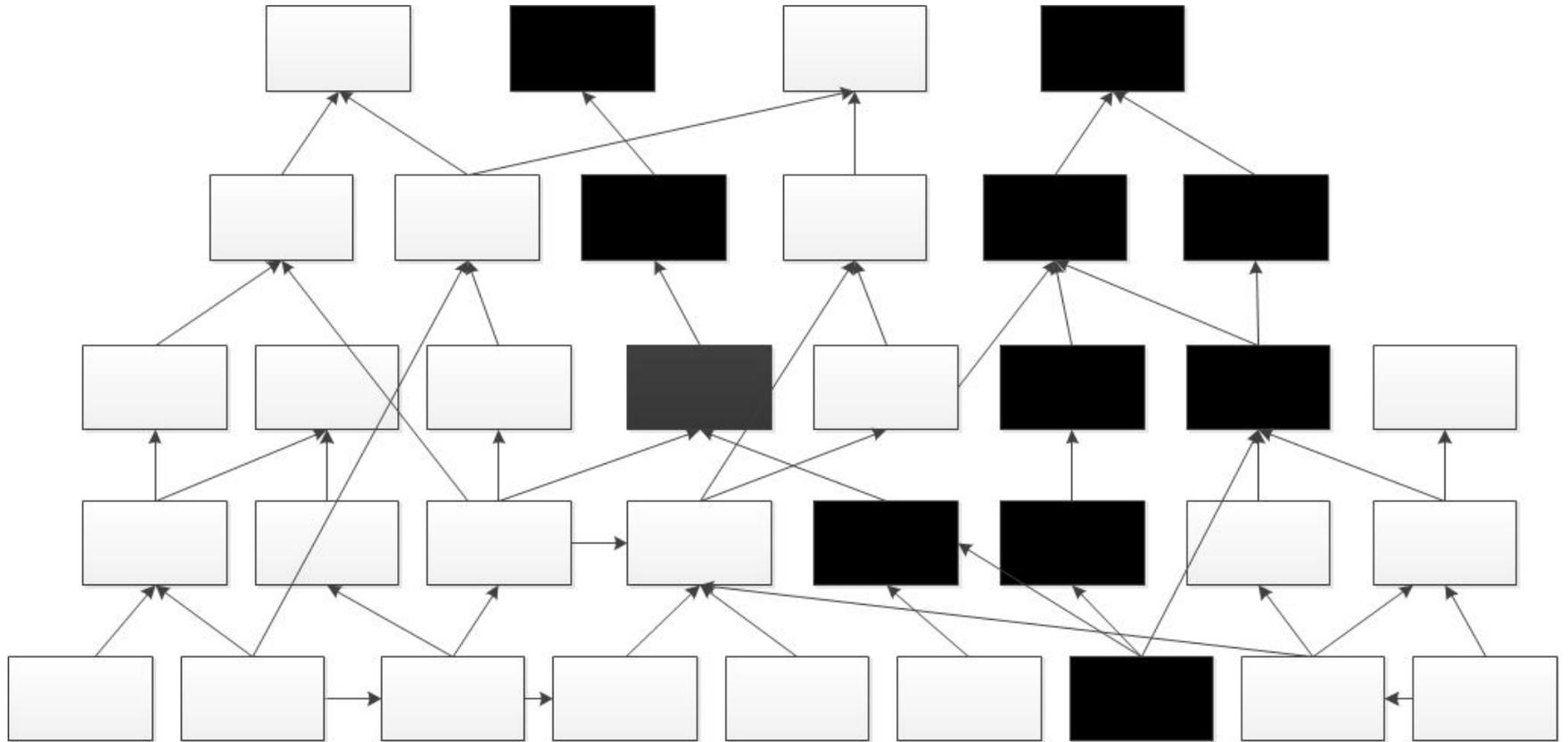
- Тестов должно быть много
- Маленькая зона видимости
- Требовательны к качеству кода

В старом проекте, не адаптированном под тестирование, написать продуктивные тесты на старый функционал без серьезного рефакторинга кода затруднительно

Про связи



Внесение изменения



Давайте на примерах



Инструменты автоматизации

- Java (IntelliJ Idea)
- Maven
- Junit 5
- Selenide
- Rest assured
- Allure2
- Jmeter
- Git, mercurial
- Bamboo

Конфигурация ролт

The screenshot shows the IntelliJ IDEA interface with the 'cmsgatesystemtests' project open. The 'pom.xml' file is selected in the Project view on the left. The main editor displays the XML content of the pom.xml file, which lists several dependencies. Red arrows highlight the following dependencies:

- `<dependency>` (lines 20-24): `io.rest-assured` with version `3.1.0` and scope `test`.
- `<dependency>` (lines 28-32): `org.junit.jupiter:junit-jupiter-api` with version `5.2.0` and scope `test`.
- `<dependency>` (lines 36-40): `org.junit.jupiter:junit-jupiter-engine` with version `5.2.0` and scope `test`.
- `<dependency>` (lines 44-48): `org.junit.platform:junit-platform-launcher` with version `1.2.0` and scope `test`.
- `<dependency>` (lines 52-56): `io.gameta.allure:allure-junit5` with version `2.7.0` and scope `test`.
- `<dependency>` (lines 60-64): `com.codeborne:selenide` with version `4.12.3` and scope `test`.

The bottom status bar shows the current directory path: `project > build > plugins > plugin`. At the very bottom, there is a notification: "IDE and Plugin Updates: The following components are ready to update: Android Emulator, Intel x86 Emulator Accelerator (HAXM installer), Android SDK Platform-Tools, Intel x86 Atom System Image (today 18:16)".

Аннотации junit 5

Annotation	Description
 @Test	Denotes that a method is a test method. Unlike JUnit 4's <code>@Test</code> annotation, this annotation does not declare any attributes, since test extensions in JUnit Jupiter operate based on their own dedicated annotations. Such methods are <i>inherited</i> unless they are <i>overridden</i> .
@ParameterizedTest	Denotes that a method is a parameterized test . Such methods are <i>inherited</i> unless they are <i>overridden</i> .
@RepeatedTest	Denotes that a method is a test template for a repeated test . Such methods are <i>inherited</i> unless they are <i>overridden</i> .
@TestFactory	Denotes that a method is a test factory for dynamic tests . Such methods are <i>inherited</i> unless they are <i>overridden</i> .
@TestInstance	Used to configure the test instance lifecycle for the annotated test class. Such annotations are <i>inherited</i> .
@TestTemplate	Denotes that a method is a template for test cases designed to be invoked multiple times depending on the number of invocation contexts returned by the registered providers . Such methods are <i>inherited</i> unless they are <i>overridden</i> .
 @DisplayName	Declares a custom display name for the test class or test method. Such annotations are not <i>inherited</i> .
 @BeforeEach	Denotes that the annotated method should be executed <i>before each</i> <code>@Test</code> , <code>@RepeatedTest</code> , <code>@ParameterizedTest</code> , or <code>@TestFactory</code> method in the current class; analogous to JUnit 4's <code>@Before</code> . Such methods are <i>inherited</i> unless they are <i>overridden</i> .
@AfterEach	Denotes that the annotated method should be executed <i>after each</i> <code>@Test</code> , <code>@RepeatedTest</code> , <code>@ParameterizedTest</code> , or <code>@TestFactory</code> method in the current class; analogous to JUnit 4's <code>@After</code> . Such methods are <i>inherited</i> unless they are <i>overridden</i> .
 @BeforeAll	Denotes that the annotated method should be executed <i>before all</i> <code>@Test</code> , <code>@RepeatedTest</code> , <code>@ParameterizedTest</code> , and <code>@TestFactory</code> methods in the current class; analogous to JUnit 4's <code>@BeforeClass</code> . Such methods are <i>inherited</i> (unless they are <i>hidden</i> or <i>overridden</i>) and must be <i>static</i> (unless the "per-class" test instance lifecycle is used).
@AfterAll	Denotes that the annotated method should be executed <i>after all</i> <code>@Test</code> , <code>@RepeatedTest</code> , <code>@ParameterizedTest</code> , and <code>@TestFactory</code> methods in the current class; analogous to JUnit 4's <code>@AfterClass</code> . Such methods are <i>inherited</i> (unless they are <i>hidden</i> or <i>overridden</i>) and must be <i>static</i> (unless the "per-class" test instance lifecycle is used).
@Nested	Denotes that the annotated class is a nested, non-static test class. <code>@BeforeAll</code> and <code>@AfterAll</code> methods cannot be used directly in a <code>@Nested</code> test class unless the "per-class" test instance lifecycle is used. Such annotations are not <i>inherited</i> .
 @Tag	Used to declare <i>tags</i> for filtering tests, either at the class or method level; analogous to test groups in TestNG or Categories in JUnit 4. Such annotations are <i>inherited</i> at the class level but not at the method level.
@Disabled	Used to <i>disable</i> a test class or test method; analogous to JUnit 4's <code>@Ignore</code> . Such annotations are not <i>inherited</i> .

Rest Assured

```
public class SwapiTest {  
    @Test  
    public void shouldGetLuke() {  
        get("http://swapi.co/api/people/1/")  
        .then().statusCode(200)  
            .and()  
            .assertThat().  
            body("name", equalTo("Luke Skywalker")); }  
}
```

Selenide

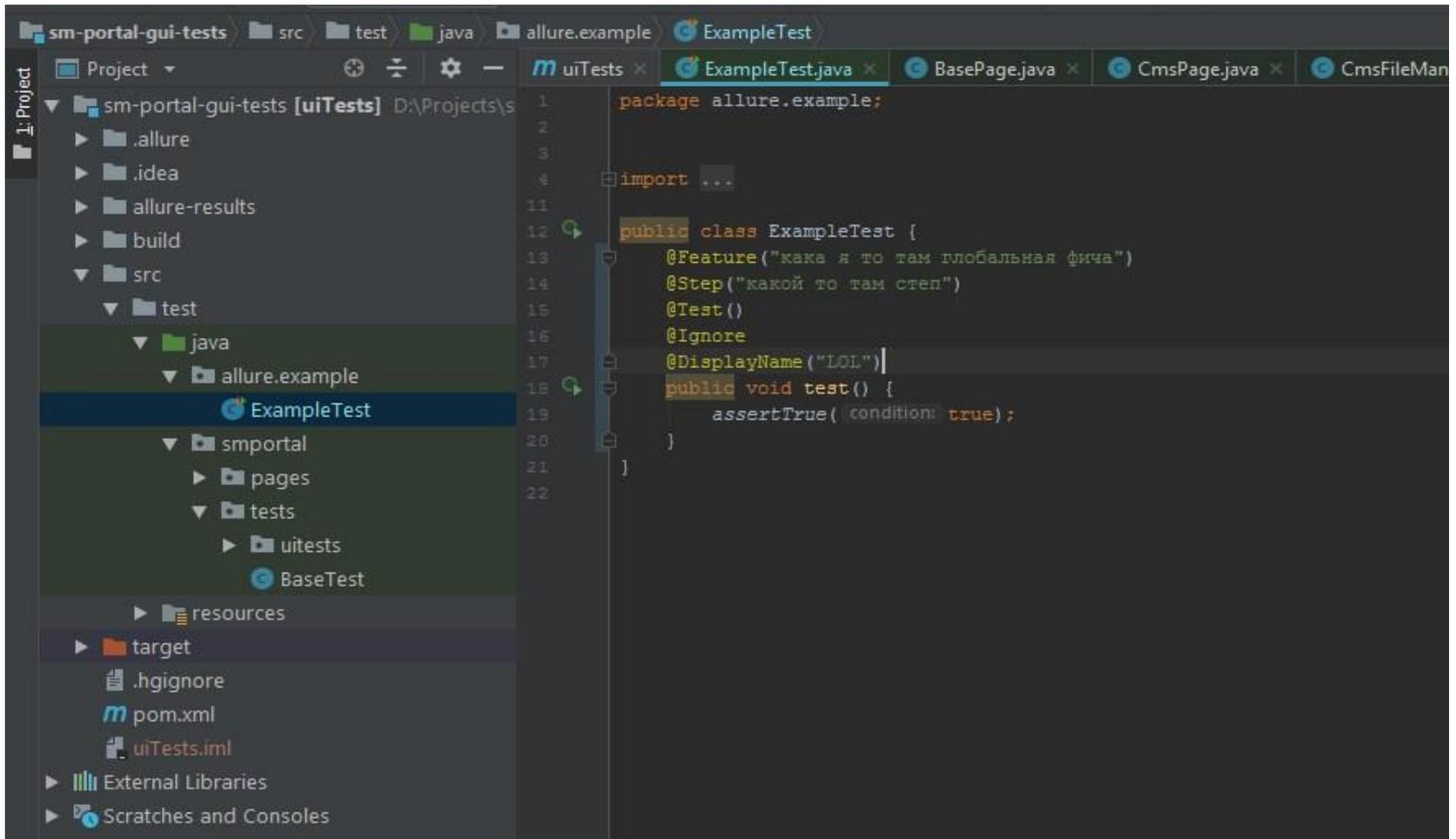
```
package ru.artlebedev.test;

import ...

public class SelenideTest extends BaseTest{

    @Test
    public void test() {
        open( relativeOrAbsoluteUrl: "https://www.artlebedev.ru");
        $( cssSelector: "[href=\\\"\\/tools\\/\\/\\\"]").click();
        $( cssSelector: "#item-matrix").click();
        $( cssSelector: "[name=\\\"Word\\\"]").setValue("блок");
        $( cssSelector: "button").click();
        $$ ( cssSelector: ".text-double-margin").shouldHave(CollectionCondition.texts("блок"));
        Assert.assertTrue( condition: false);
    }
}
```

allure



The screenshot shows an IDE window for a project named 'sm-portal-gui-tests'. The left sidebar displays the project structure, including folders for '.allure', '.idea', 'allure-results', 'build', 'src', 'target', and files like 'pom.xml' and 'uiTests.iml'. The main editor area shows the code for 'ExampleTest.java' in the 'allure.example' package. The code includes several Allure annotations: @Feature, @Step, @Test, @Ignore, and @DisplayName. The test method 'test()' contains a single assertion: 'assertTrue(condition: true);'.

```
1 package allure.example;
2
3
4 import ...
5
6
7
8
9
10
11
12 public class ExampleTest {
13     @Feature("кака я то там глобальная фича")
14     @Step("какой то там шаг")
15     @Test()
16     @Ignore
17     @DisplayName("LOL")
18     public void test() {
19         assertTrue(condition: true);
20     }
21 }
22
```

Подключение allure

```
<plugin>
  <groupId>io.qameta.allure</groupId>
  <artifactId>allure-maven</artifactId>
  <version>2.9</version>
  <configuration>
    <resultsDirectory>${basedir}/target/allure-results</resultsDirectory>
    <reportDirectory>${basedir}/target/allurereport</reportDirectory>
  </configuration>
</plugin>
```

Подключение тестов к CI

The screenshot shows the Bamboo CI dashboard for a plan named "test-demix-uat-gui". The top navigation bar includes "Bamboo", "My Bamboo", "Projects", "Build", "Deploy", "Reports", and "Create". A search bar and user profile are also visible. The main content area is divided into several sections:

- Plan summary**: Includes tabs for "Recent failures", "History", "Tests", and "Issues".
- Current activity**: States "No builds are currently running."
- Recent history**: A table listing the last 10 builds, all of which failed.
- Plan statistics**: Shows "25 builds", "0% successful", and "1m average duration". It includes a pie chart showing 0% success and a bar chart showing build durations.

Below the recent history table, there are links for "Latest build" and "Feed for all builds or just the failed builds."

Build ID	Triggered by	Time ago	Status
#36	Triggered by Deployment for demix > dmkuat_front	3 hours ago	18 of 22 failed
#35	Triggered by Deployment for demix > dmkuat_front	4 hours ago	18 of 22 failed
#34	Triggered by Deployment for demix > dmkuat_front	5 hours ago	18 of 22 failed
#33	Triggered by Deployment for demix > dmkuat_front	5 hours ago	18 of 22 failed
#32	Triggered by Deployment for demix > dmkuat_front	5 hours ago	18 of 22 failed
#31	Triggered by Deployment for demix > dmkuat_front	1 day ago	18 of 22 failed
#30	Triggered by Deployment for demix > dmkuat_front	1 day ago	18 of 22 failed
#29	Triggered by Deployment for demix > dmkuat_front	1 day ago	18 of 22 failed
#28	Triggered by Deployment for demix > dmkuat_front	1 day ago	18 of 22 failed
#27	Triggered by Deployment for demix > dmkuat_front	1 day ago	18 of 22 failed

Отчет allure в Bamboo, анализ результатов

ⓘ #37 failed – Manual run by Чибисов Денис Владимирович

Summary Tests Allure Report Commits Artifacts Logs Metadata Issues



Overview
Categories
Suites
Graphs
Timeline
Behaviors
Packages

Behaviors

order name duration status

Filter by status: 5 0 17 0 0

- Баннеры на главной
 - Тестируем переходы по ссылкам на банерах
 - #5 проверяем на главной переход по всем банерам с видами спорта 54s 439ms
 - #7 проверяем переход с баннера на страницу регистрации 4m 14s
 - #6 проверяем переходы по основным категориям на банерах ОДЕЖДА/ОБУВЬ/АКСЕССУАРЫ/ИНВЕНТАРЬ 18s 871ms
 - #3 проверяем переходы по ссылкам на банере с обувью 10s 709ms
 - #4 проверяем переходы по ссылкам на банере с одеждой 14s 016ms
 - #1 проверяем прокрутку банеров и наличие кнопок на них 7s 845ms
 - #2 Проверяем что картинки отображаются(футболка, кроссовок, рюкзаки, мяч) 1s 437ms
 - Геолокация 1
 - Демикс клуб 2
 - Карточка товара 1
 - Каталог товаров 1
 - Корзина 1

>	Запоминаем название категории 1 parameter	15ms
>	Нажимаем на банере категорию 1 parameter	2s 198ms
>	Проверяем что текст на банере ОДЕЖДА 1 parameter	154ms
✔	Открываем главную страницу	1s 539ms
>	Проверяем что основные категории на банерах существуют 1 parameter	5ms
>	Запоминаем название категории 1 parameter	19ms
>	Нажимаем на банере категорию 1 parameter	3s 199ms
✔	Проверяем что текст на банере ОБУВЬ 1 parameter imageName ОБУВЬ	28ms
✔	Открываем главную страницу	2s 122ms
>	Проверяем что основные категории на банерах существуют 1 parameter	73ms
>	Запоминаем название категории 1 parameter	14ms
>	Нажимаем на банере категорию 1 parameter	3s 580ms
>	Проверяем что текст на банере АКСЕССУАРЫ 1 parameter	81ms
✔	Открываем главную страницу	2s 486ms
>	Проверяем что основные категории на банерах существуют 1 parameter	30ms
✔	Нажимаем на _инвентарь_ на банере	696ms
✔	Проверяем правильность текущего urlа	14ms

Введение регламентов ИСПОЛЬЗОВАНИЯ

Summary Tests Allure Report Commits Artifacts Logs Metadata Issues

Allure

- Overview
- Categories
- Suites
- Graphs
- Timeline
- Behaviors
- Packages

ALLURE REPORT 08/08/2018

13:14:57 - 13:15:23 (26s 295ms)

21
test cases

85.71%

SUITES 21 items total

portal.tests.file_browser.UploadImageTest	1
portal.tests.file_browser.CreateFolderTest	1
portal.tests.navigation.NavigationCreateInternalPageTest	1
portal.tests.widgets.WidgetsCreateAndShowTest	1
portal.tests.knowledge.base.CreateFormsKnowledgeItemTest	1
portal.tests.knowledge.base.CreateInstructionsKnowledgeItemTest	1
portal.tests.knowledge.base.StandardsKnowledgeItemTest	1
portal.tests.knowledge.base.CreateOthersKnowledgeItemTest	1
portal.tests.requests.change_permit.ChangePermitRequestChangeResponseAndSendTest	1
portal.tests.interviews.InterviewCreateAndNotStartYetTest	1

TREND

Build	Pass	Fail	Unknown
#21	17	0	0
#22	17	0	0
#23	17	0	0
#24	14	0	0
#25	14	0	0
#26	14	0	0
#27	14	0	0
#28	14	0	0
#29	14	0	0
#30	14	0	0
#31	14	0	0
#32	14	0	0
#33	14	0	0
#34	14	0	0
#35	14	0	0
#36	14	0	0
#37	14	0	0
#38	14	0	0
#39	14	0	0
#40	14	0	0

CATEGORIES 1 item total

Product defects	1
-----------------	---

Show all

EXECUTORS

Bamboo sm-portal-rests-tests-dev

2373_42.jpg Без названия.jpg speech-bubbles-wi...zip Показать все

А точно надо?

Без внедрения общей методологии тестирования невозможно:

- обеспечить полное соответствие программ поставленным функциональным требованиям
- обеспечить высокий уровень надежности работы программного обеспечения
- добиться стабильно высокого качества программного обеспечения

Без тестирования

