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

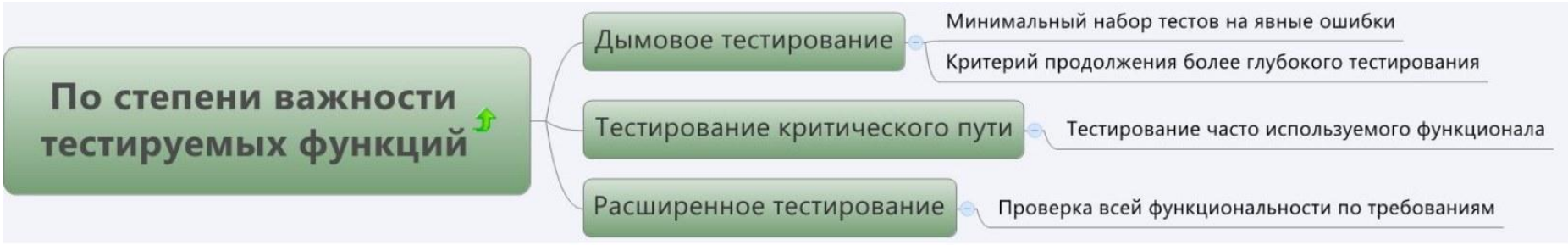
ГОСТ Р ИСО/МЭК 9126-93. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению

Классификация видов и направлений тестирования

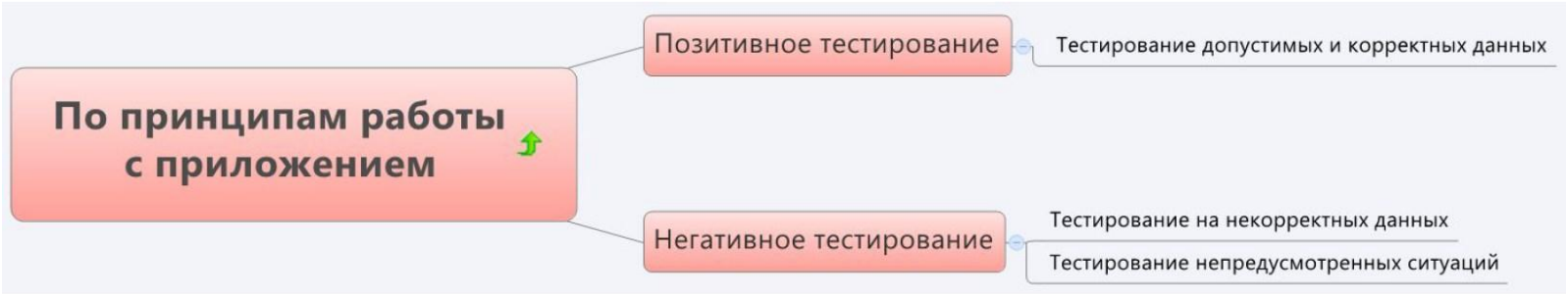
По запуску кода на исполнение

По доступу к коду и архитектуре приложения

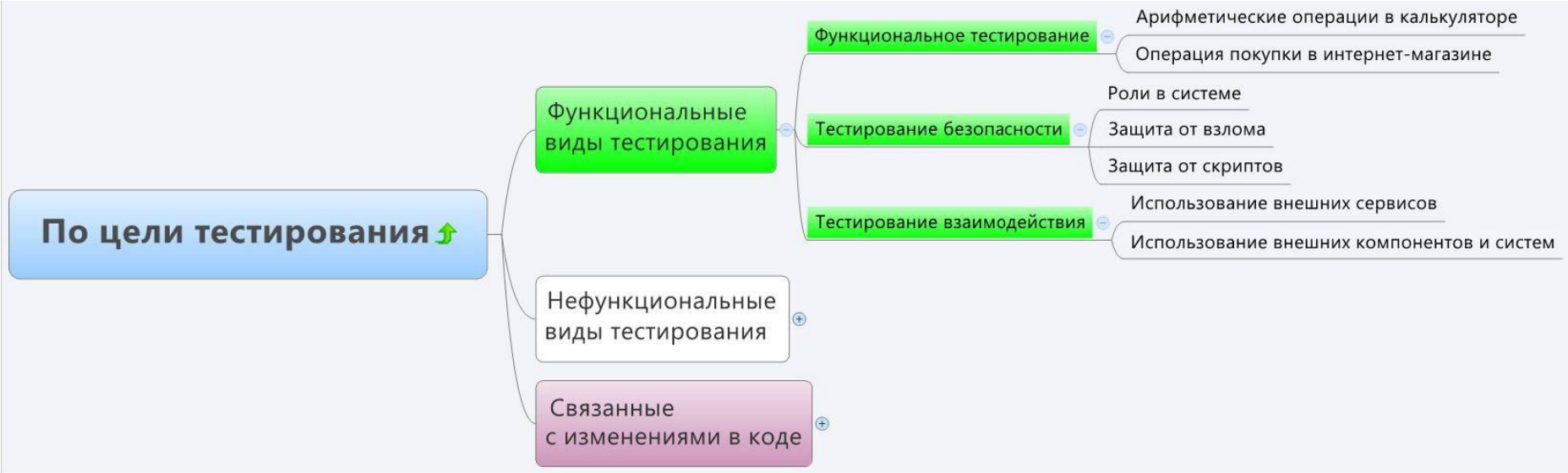
По степени важности тестируемых функций



По принципам работы с приложением



По целитестирования



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

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

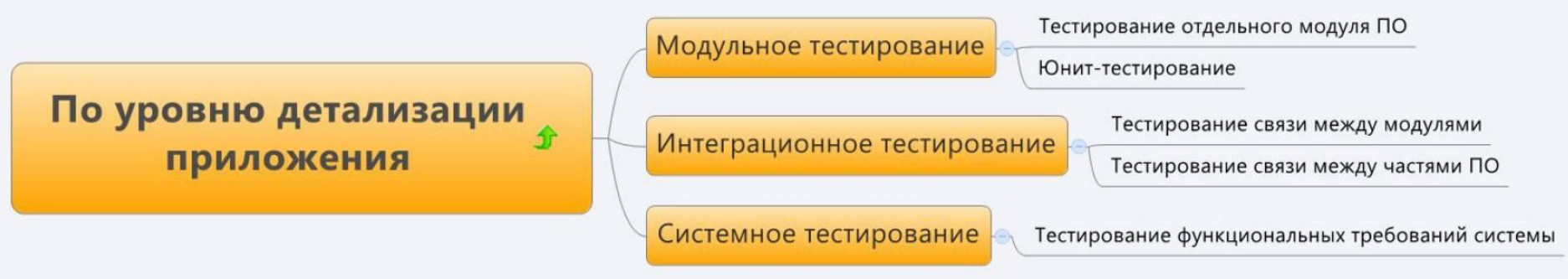
Что это?

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)



Из чего состоит?

Системное тестирование

Это тестирование программного обеспечения
,
выполняемое на полной, интегрированной систе
ме,
с целью проверки соответствия системы исходн
ым 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

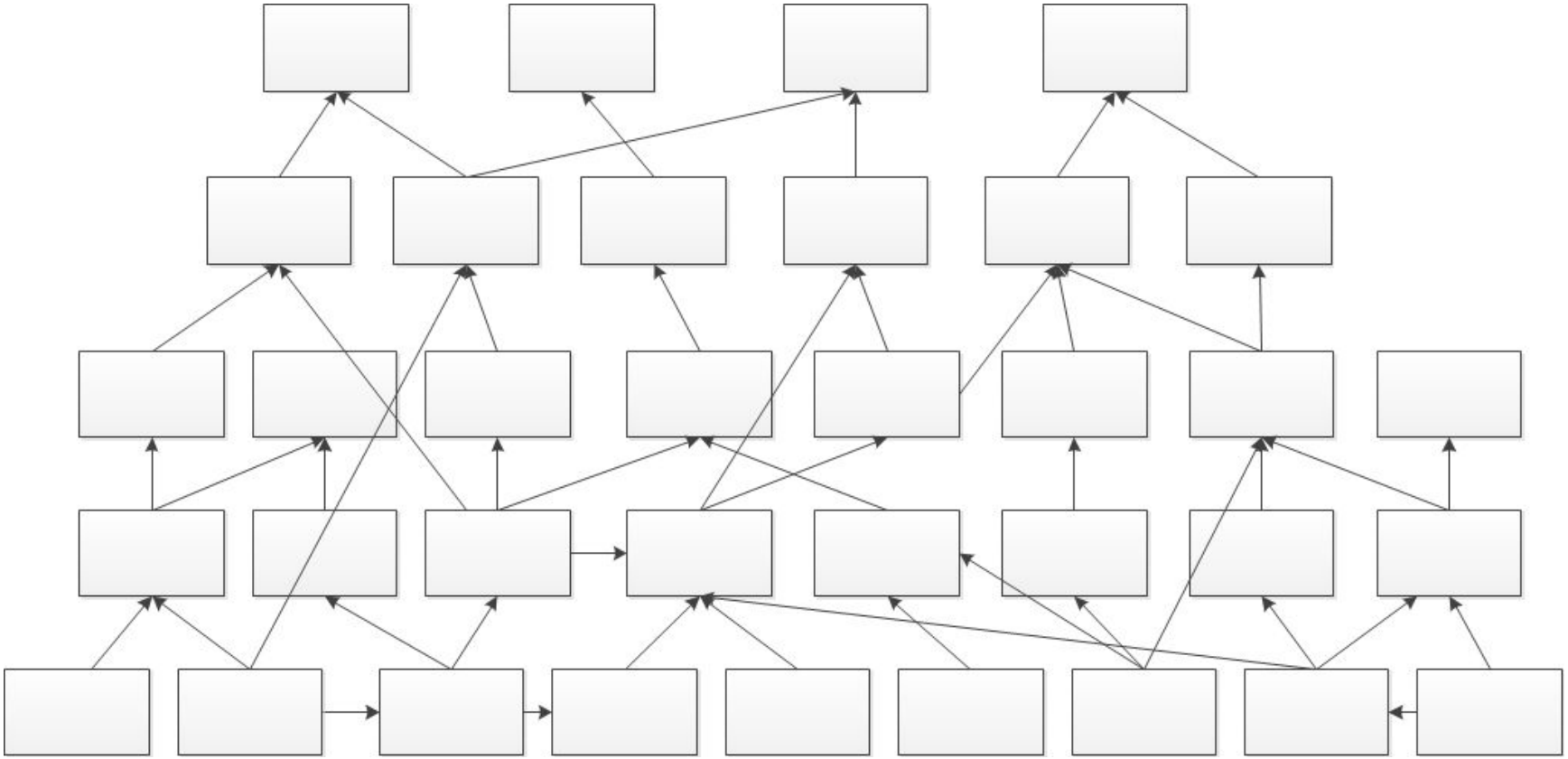
- Легко пишутся
- Высокая локализация ошибки
- Быстро выполняются
- Легко поддерживать
- Повышают качество кода программы

Минусы модульного тестирования

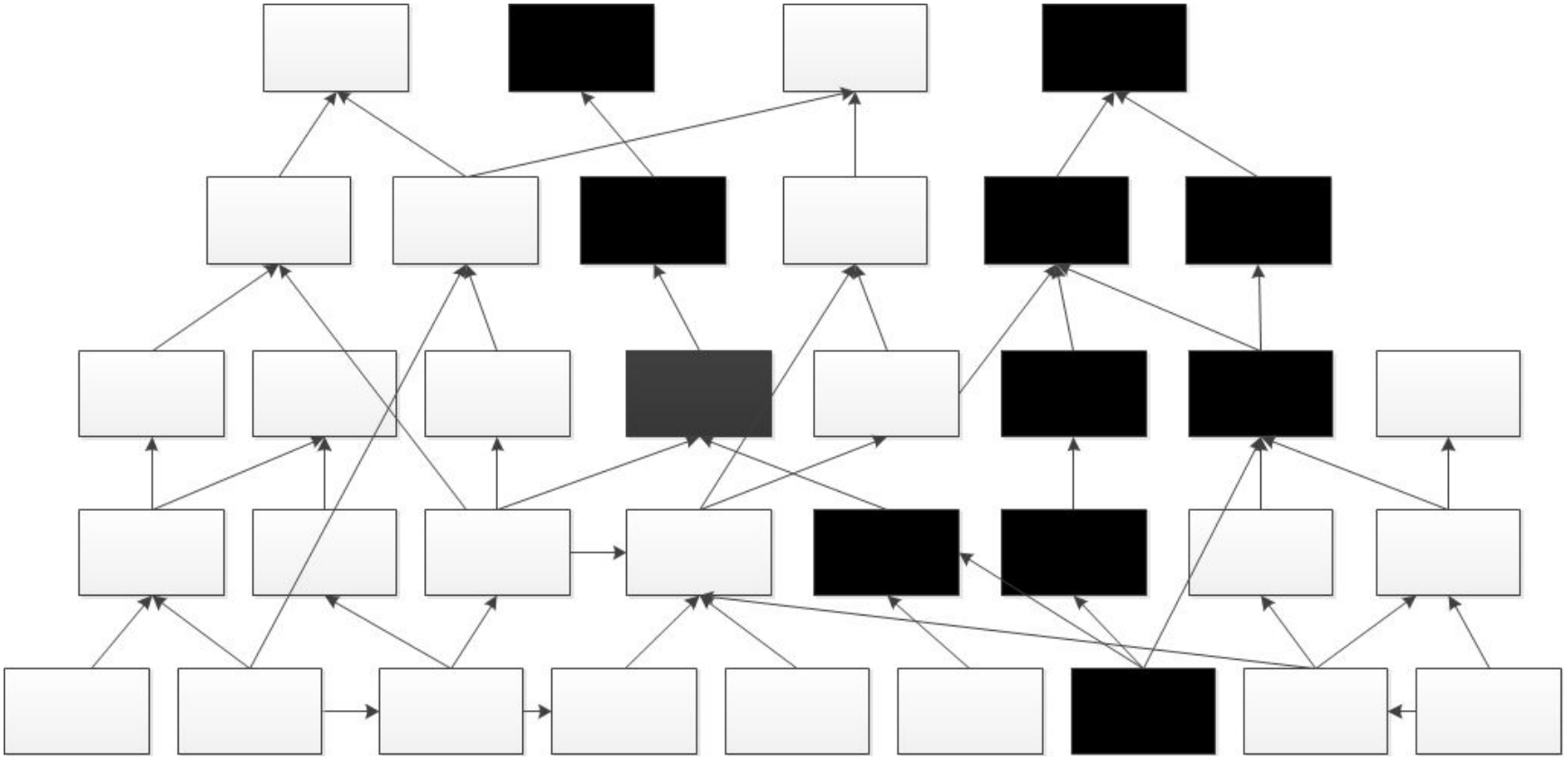
- Тестов должно быть много
- Маленькая зона видимости
- Требовательны к качеству кода

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

Про связи



Внесение изменения

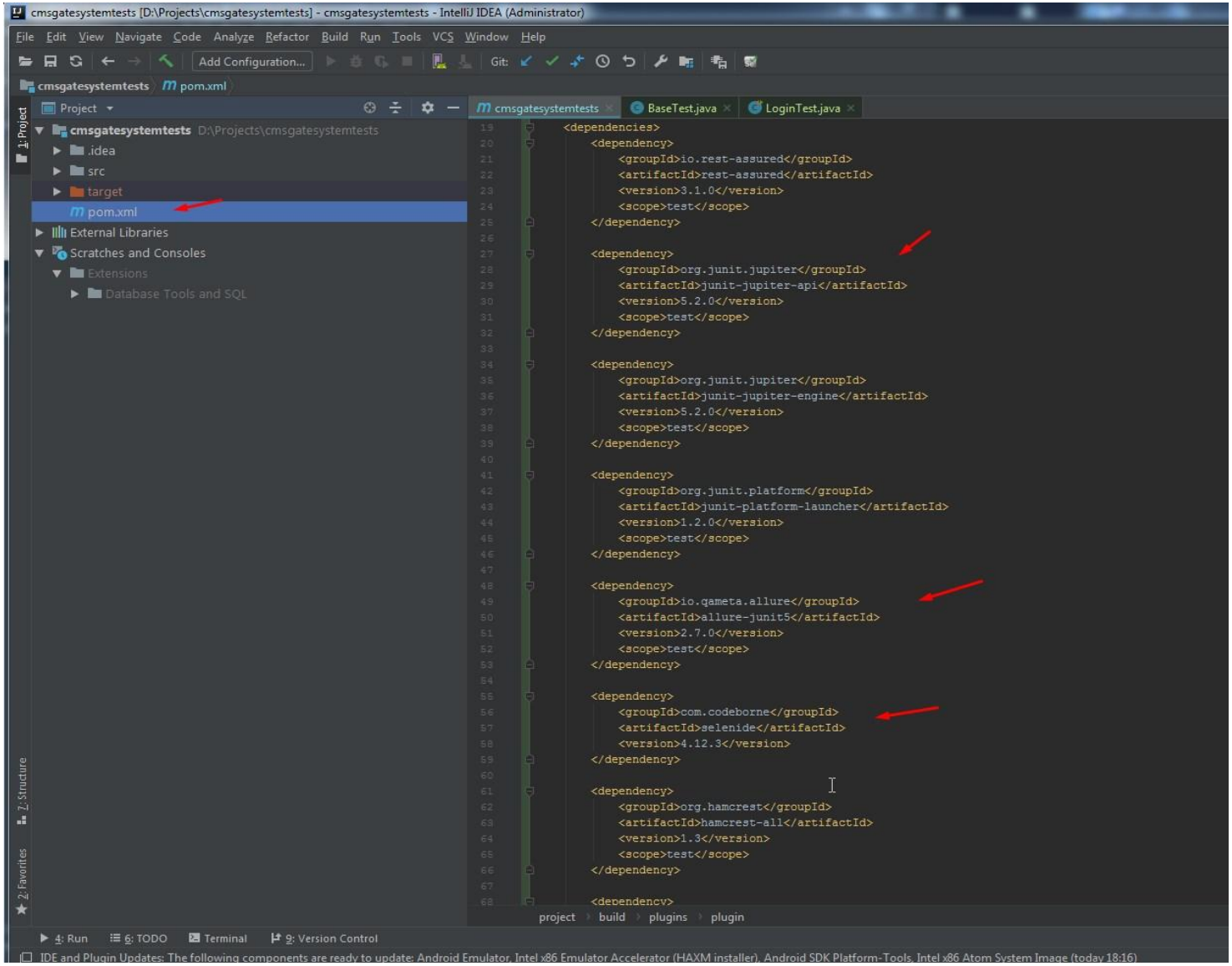


Давайте на примерах

Инструменты автоматизации

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

Конфигурация pom



Аннотации junit 5

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

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

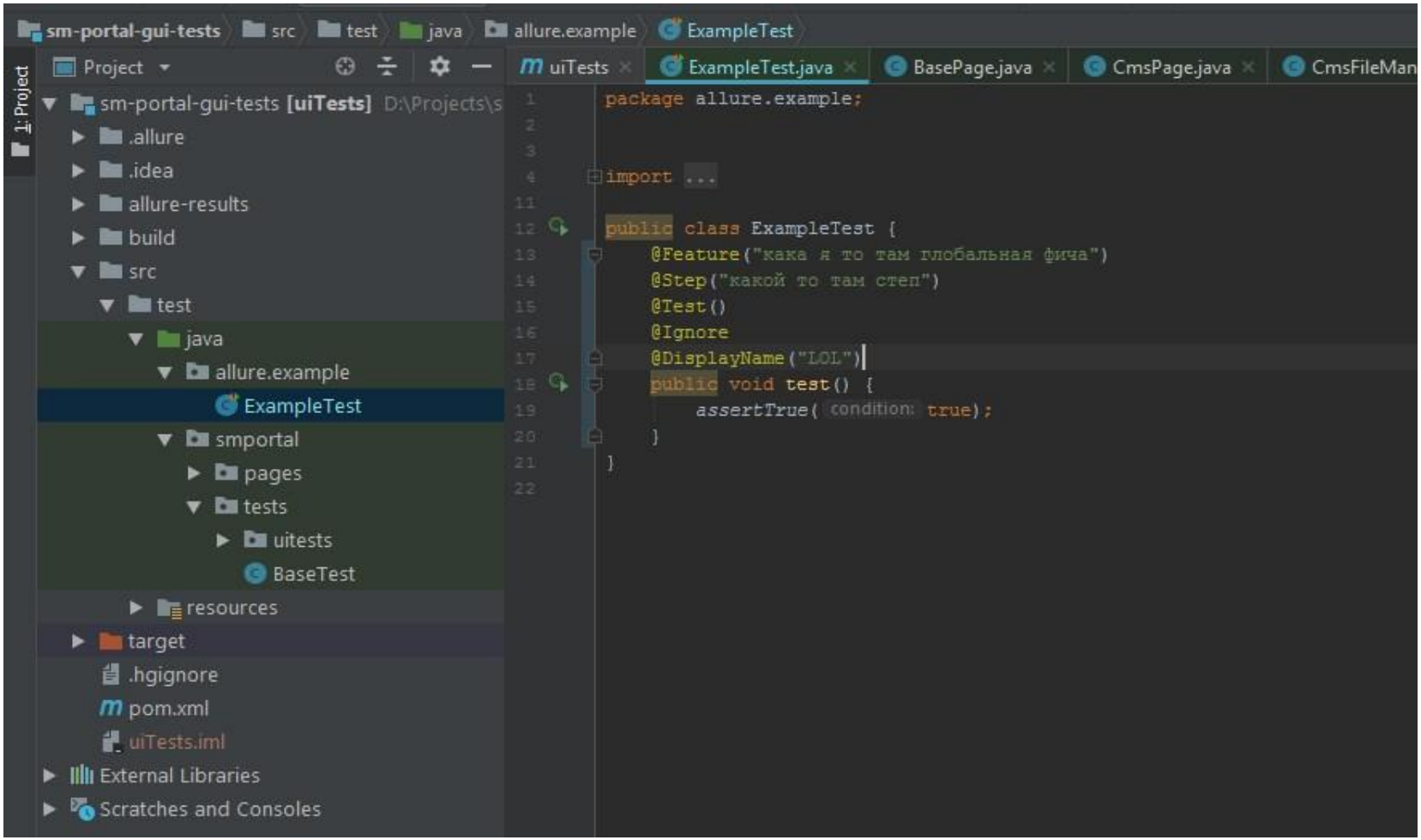
```java
package ru.artlebedev.test;

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);
    }
}
```

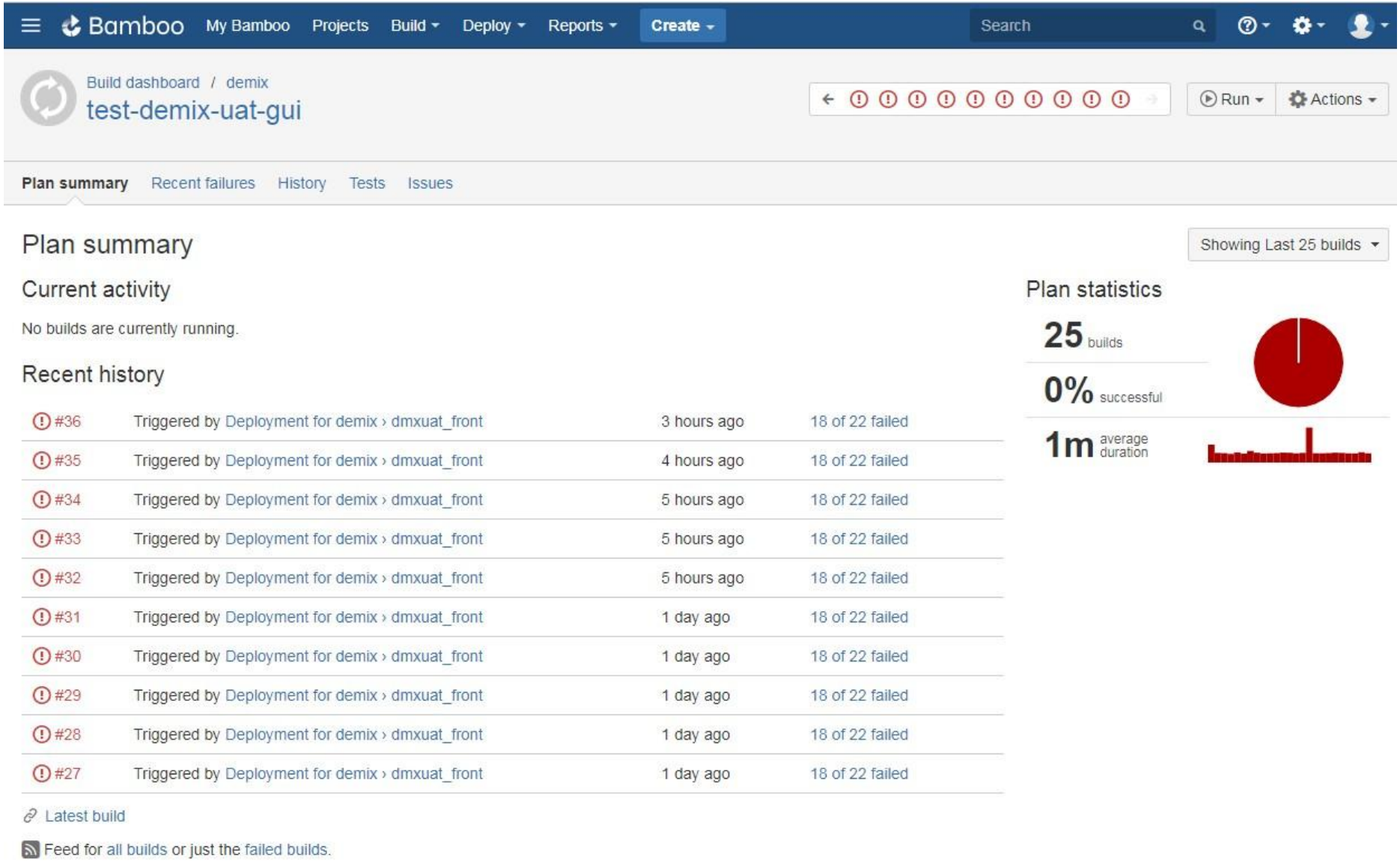
allure


sm-portal-gui-tests
src
test
java
allure.example
ExampleTest
Project
sm-portal-gui-tests [uiTests] D:\Projects\s
.allure
.idea
allure-results
build
src
test
java
allure.example
ExampleTest
smportal
pages
tests
uitests
BaseTest
resources
target
.hgignore
pom.xml
uiTests.iml
External Libraries
Scratches and Consoles
uiTests
ExampleTest.java
BasePage.java
CmsPage.java
CmsFileMan
package allure.example;
import ...
public class ExampleTest {
@Feature("кака я то там глобальная фича")
@Step("какой то там степ")
@Test()
@Ignore
@DisplayName("LOL")
public void test() {
assertTrue(condition: true);
}
}

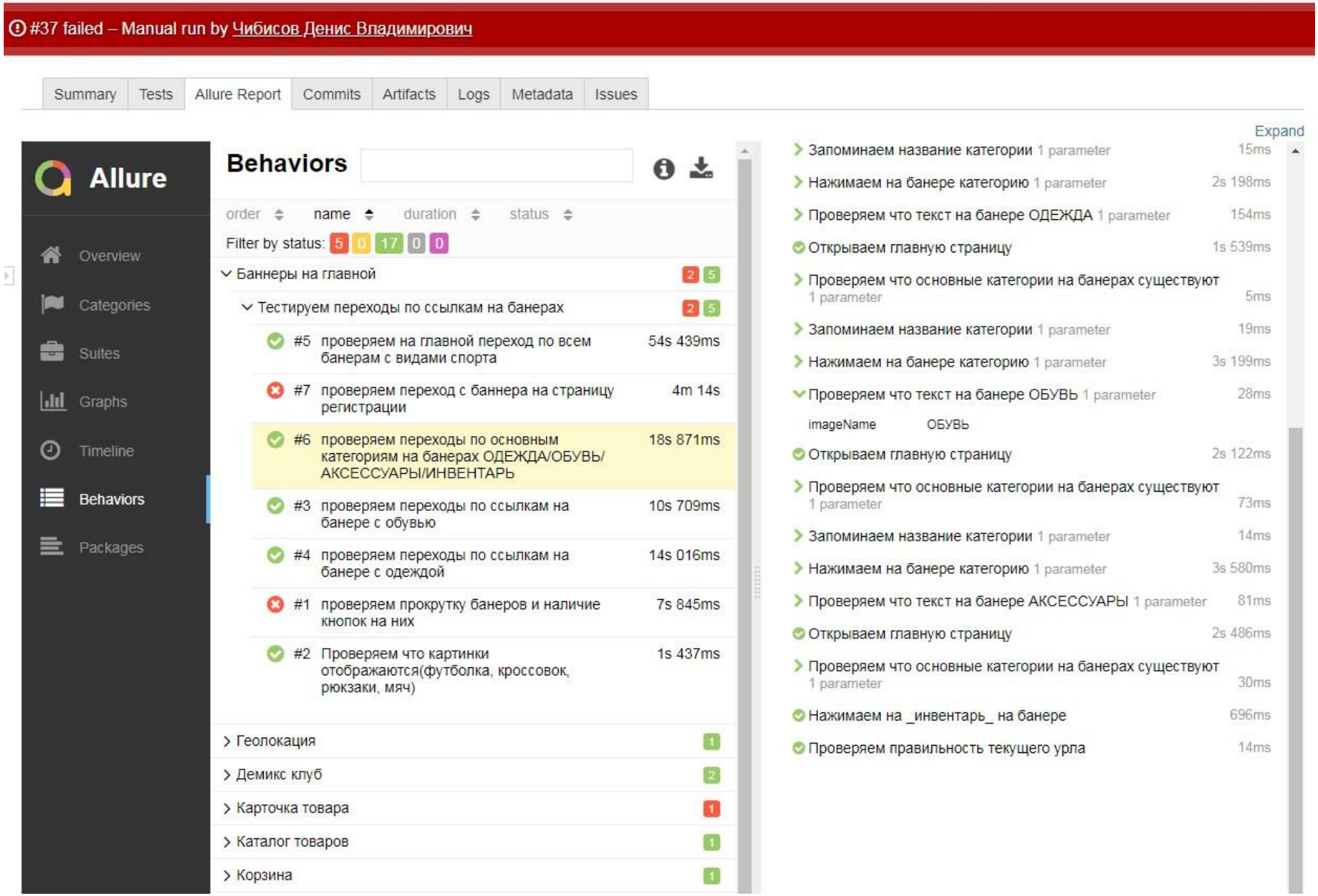
Подключение allure

```
<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>
```

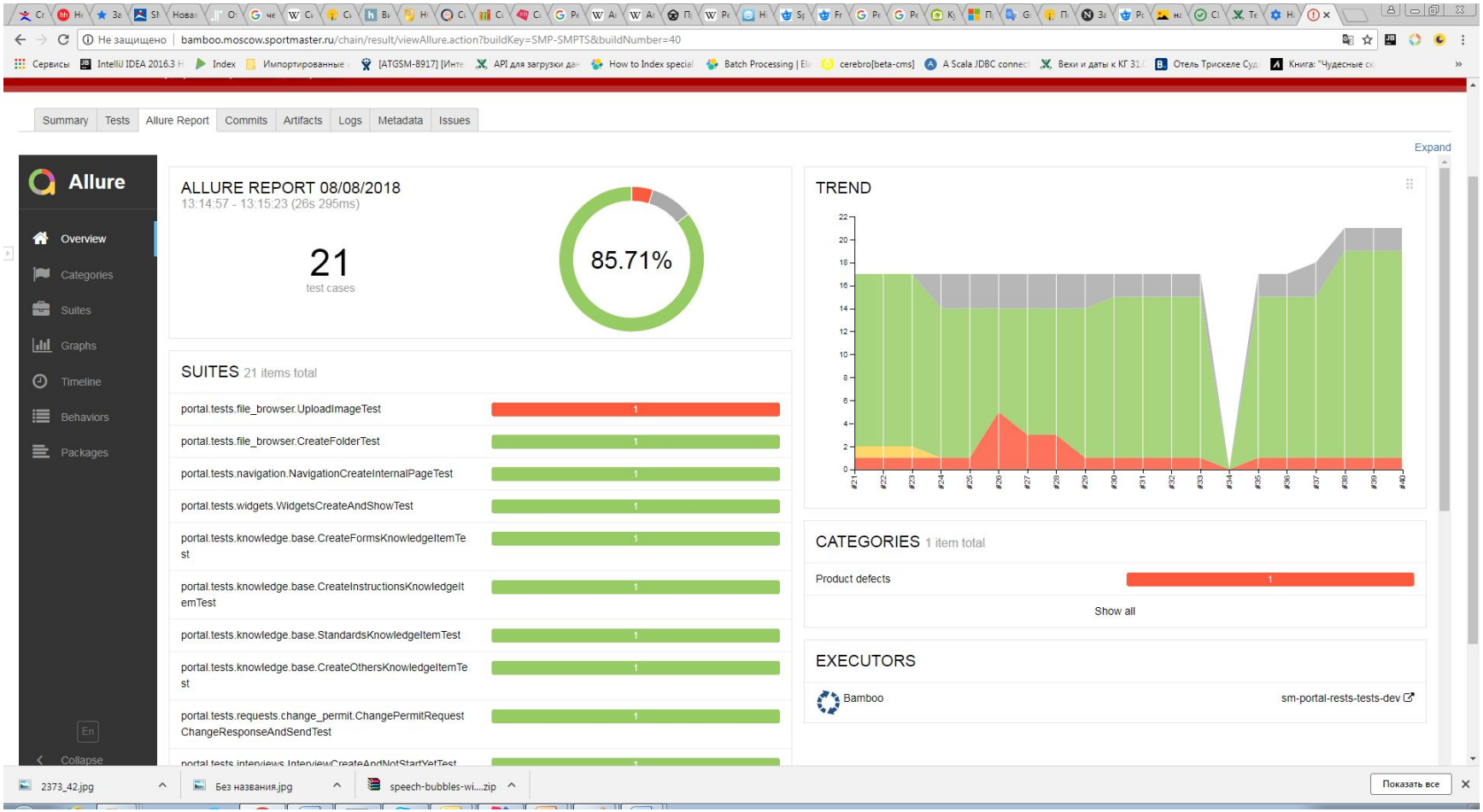
Подключение тестов к CI



Отчет allure в Bamboo, анализ результатов



Введение регламентов использования



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