Санкт-Петербургский государственный университет

Кафедра системного программирования

Анализ нестабильности выделенного юнит-теста

Демченко Артем Евгеньевич, группа 22.Б11-мм

Научный руководитель: к. ф.-м. н., доцент Куликов Е. К.

Санкт-Петербург
2025

Введение

- Хотя бы один тест упал — сборка считается неуспешной
- Нестабильные тесты — тесты, результат которых может меняться без изменения исходного кода
- 1,5% всех тестов Google — нестабильные[1]
- Наличие в проекте нестабильных тестов замедляет процесс разработки
- Возникает потребность в инструментах, анализирующих тесты на нестабильность
- Explyt Test — плагин для IntelliJ IDEA, динамично развивающийся продукт
  - Сейчас — генератор тестов для Java и Kotlin, AI-ассистент
  - В будущем — инструмент всестороннего ухода за тестами

[1] По данным из Google testing blog на 2016 год

Постановка задачи

Цель работы — разработать анализатор, определяющий, является ли выделенный юнит-тест нестабильным, и предоставляющий полезную для исправления информацию. Для достижения цели были поставлены задачи:

Задачи:

- Рассмотреть существующие способы анализа нестабильности тестов
- Реализовать анализатор юнит-теста на предмет нестабильности, извлекающий полезную для его исправления информацию
- Исследовать возможности больших языковых моделей в анализе причин нестабильности тестов
- Апробировать разработанный инструмент на наборе модельных примеров и на нестабильных тестах из реальных проектов
- Реализовать механизм представления результатов анализа пользователю
- Внедрить разработанные компоненты в Explyt Test

Методы обнаружения нестабильных тестов

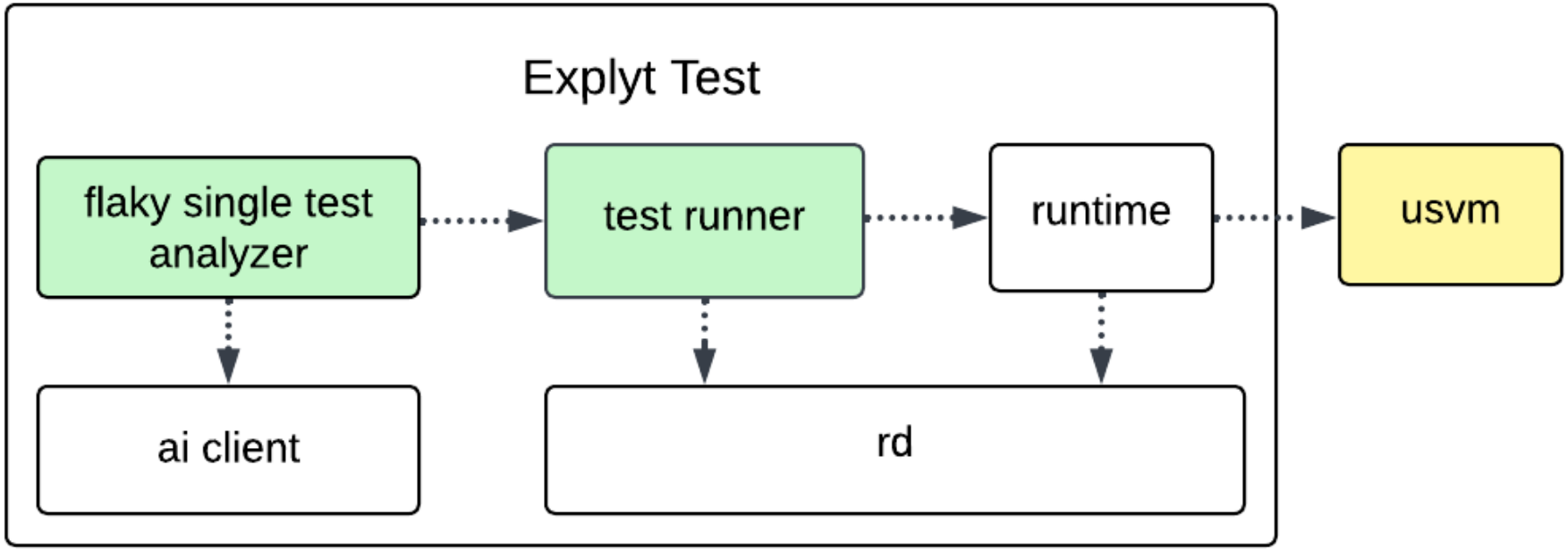
A Survey of Flaky Tests — Owain Parry, Gregory M. Kapfhammer, Michael Hilton, Phil McMinn, 2021, https://doi.org/10.1145/3476105

- iDFlakies
  - Maven-плагин, работающий только с тестовым фреймворком JUnit
  - Используется стратегия ReRun — запуск тестов происходит несколько раз
- DeFlaker
  - Запуск тестов происходит один раз при внесении изменений в систему контроля версий
  - Считается покрытие кода тестами
  - Тест упал и измененный код не покрыт — тест помечается как нестабильный
  - Нацелен на обнаружение нестабильных тестов, которые уже существуют в проекте

Методы обнаружения нестабильных тестов

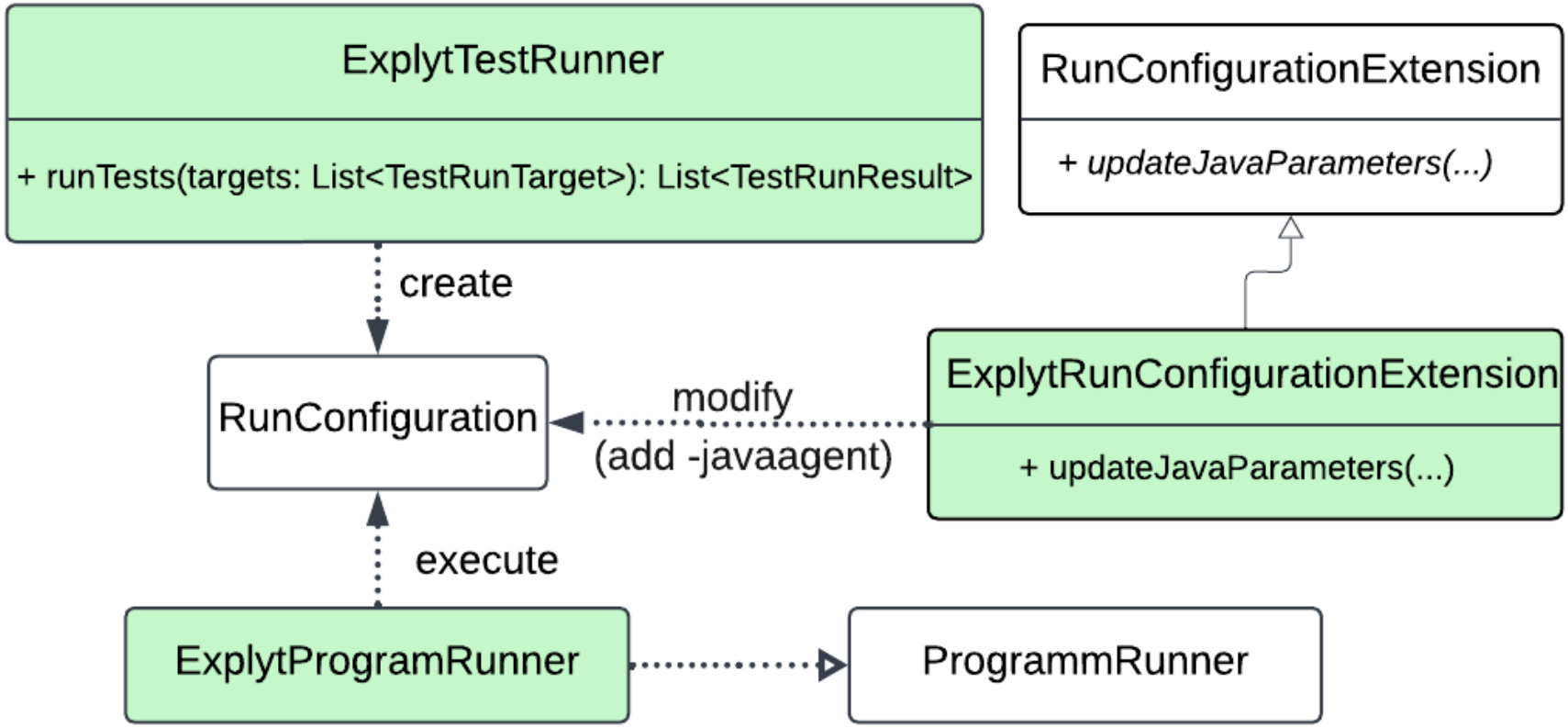
- NonDex
  - Поиск неверных предположений о детерминированности недетерминированных спецификаций в Java
  - Пример — порядок итерации по элементам HashMap
  - Тесты запускаются с инструментацией, подменяя реализацию недетерменированного API
  - Поддержка 41 спецификаций
- Flakify
  - Подход на основе дообученной языковой модели CodeBERT
    - Модель имеет ограничение на количество входных токенов — 512
  - Инструмент не поддерживается более 2-ух лет
  - Отсутствие документации по использованию

Зависимость модулей в Explyt Test



- runtime — предоставляет java agent для инструментации
- rd — фреймворк для межпроцессного взаимодействия
- ai client — предоставляет API для взаимодействия с LLM
- usvm — добавляет возможность сбора трасс

Архитектура test runner



Алгоритм работы анализатора

Определяем нестабильность теста:

1. Запускаем пользовательский юнит-тест N раз
2. Смотрим, есть ли разные результаты выполнения тестов

Получаем информацию о нестабильности:

1. Сравнивая успешную и провальную трассы, получаем инструкцию, где исполнение расходится
2. По совпадающему префиксу трасс строим стек вызовов
   - Call-инструкции кладем на стек
   - Встречаем return — убираем со стека последнюю добавленную
3. Сопоставляем каждой инструкции строчку в исходном файле

Отправляем запрос в LLM, используя ai client

Пример работы на модельном примере

```java
public class FlakyTests {

    @Test
    void randomPass() {
        doTest();
    }

    void doTest() {  1 usage
        int r = new Random().nextInt( bound: 2);
        Assertions.assertEquals( expected: 0, r);
    }
}
```

Flakiness Analysis Result

Test is flaky
Common stack trace before difference in execution:
---
flaky.FlakyTests.randomPass(FlakyTests.java:11)
flaky.FlakyTests.doTest(FlakyTests.java:12)

The line after which the execution differs:
Assertions.assertEquals(0, r); | in file FlakyTests.java

Explyt Plugin Copy

Explain reason why this test is flaky randomPass()

Attachments Reuse

Explyt AI Copy

Bugs Explanation and Correction

1. **Description**: The test method `randomPass()` is designed to call `doTest()`, which generates a random integer (`r`) between 0 and 1. The assertion checks if `r` is equal to 0. This creates a flaky test because the outcome of the test is non-deterministic; it can pass or fail based on the random value generated. This violates the principle of unit tests being deterministic and reliable.
2. **Trigger Conditions**: The bug manifests every time the test is run. Since `new Random().nextInt(2)` can return either 0 or 1, the test will fail approximately 50% of the time, leading to inconsistent test results. This unpredictability can cause confusion and make it difficult to ascertain the correctness of the code being tested.
3. **Fix Suggestions**: To correct this issue, the test should be modified to remove the randomness. Instead of relying on a random value, the test should assert a known condition or behavior. For example, if the intention is to test a specific functionality, the test should use fixed inputs and expected outputs. Here's a corrected version of the test:

Решенные задачи

В ходе данной производственной практики были решены следующие задачи:

- Рассмотрены существующие способы анализа нестабильности тестов. Среди них для реализации был выбран тот, который лучше всего подходит для интеграции в Explyt Test
- Реализован анализатор, определяющий, является ли выделенный юнит-тест нестабильным, и предоставляющий данные для его исправления. Они включают первую строчку кода, после которой расходится исполнение провального и успешного запуска, и стек вызовов, приводящий к расхождению
- Реализована первая версия запроса к большой языковой модели и получение результатов
- Анализатор апробирован на модельном примере

Планы

Дальше планируется решить следующие задачи:

- Разработать специализированный промпт для взаимодействия с большой языковой моделью
- Исследовать возможности больших языковых моделей в анализе причин нестабильности тестов
- Апробировать разработанный инструмент на нестабильных тестах из реальных проектов
- Реализовать механизм представления результатов анализа пользователю
- Внедрить разработанные компоненты в Explyt Test