Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б11-мм

Анализ нестабильности юнит-тестов, вызванной порядком запуска

ШИШИН Кирилл Александрович

Отчёт по производственной практике
в форме «Производственное задание»

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

Санкт-Петербург
2025

Оглавление

1. Введение

Тестирование программного обеспечения является важной и неотъемлемой частью процесса разработки. Однако, как и в любой компоненте программного обеспечения, в тестах могут допускаться ошибки: например, некорректная работа с многопоточностью, использование нестабильных внешних сервисов, отсутствие очистки общего для тестов состояния и т.п. Это может приводить к появлению так называемых моргающих тестов — тестов, результат выполнения которых недетерминирован при отсутствии изменений в тестируемом коде. Выявление и дальнейшее устранение таких тестов отнимает много сил и времени у программистов и тестировщиков. В связи с этим последние годы активно [1] разрабатываются решения для автоматического анализа моргающих тестов. Актуальность темы подтверждается в том числе и наличием постоянно пополняемого репозитория[1], созданного специально для исследования моргающих тестов и содержащего уже более 7000 примеров таких тестов из реальных проектов (например, *Guava*[2], *Gson*[3], *Seata*[4]).

Несмотря на то количество исследований (см., например, обзорную работу [2]), которое уже было проведено, универсального инструмента, способного выявлять и устранять сразу все виды моргающих тестов, пока не существует. Большинство исследований направлено на изучение отдельных видов моргающих тестов, каждый из которых имеет свои уникальные причины возникновения и требует специального подхода к анализу [3], [4]. И среди всех этих видов, из-за чего им уделяется особое внимание, самыми распространенными в реальных проектах (согласно статистике [2], они составляют порядка 16% от всех моргающих тестов) являются порядкозависимые моргающие (ODFlaky) тесты [5], [6], [7] — тесты, результаты выполнения которых определяются последовательностью запуска тестов.

Рассмотрим минималистичный пример тестового класса, содержаще-

[1]International Dataset of Flaky Tests. – URL: https://github.com/TestingResearchIllinois/idoft

[2]Guava: Google Core Libraries for Java. – URL: https://github.com/google/guava

[3]Gson: A Java serialization/deserialization library. – URL: https://github.com/google/gson

[4]Seata: Simple Extensible Autonomous Transaction Architecture. – URL: https://github.com/apache/incubator-seata

го ODFlaky тесты,

```
class CatTest {
    private static final Cat cat = new Cat();

    @Test
    void initialNameTest() {
        String initialCatName = cat.getCatName();

        assertEquals("Cat", initialCatName);
    }

    @Test
    void catRenameTest() {
        cat.setCatName("Barsik");
        String catName = cat.getCatName();

        assertEquals("Barsik", catName);
    }
}
```

где `Cat` — представленный ниже класс, имеющий поле `catName` и соответствующие ему методы `get` и `set`.

```
class Cat {
    private String catName = "Cat";

    public String getCatName() {
        return catName;
    }

    public void setCatName(String catName) {
        this.catName = catName;
    }
}
```

Как можно видеть из результатов запусков некоторых последовательностей тестов из класса `CatTest`, представленных на рисунке 1, тест `initialNameTest` является ODFlaky, поскольку имеет разные результаты прохождения при запуске в одиночку и при последовательном запуске после `catRenameTest`. Проанализировав результаты запусков более подробно, можно дополнительно узнать, что причиной падения теста `initialNameTest` стало изменение значения поля `catName`, инициированное тестом `catRenameTest`.

Рис. 1: Результаты запусков тестов из класса `CatTest`

Информация о наборе тестов и, при возможности, о том, исполнение каких именно инструкций при выполнении этих тестов, привело к возникновению ODFlaky теста, стала бы ценным результатом для автоматического анализатора порядкозависимых тестов. А в условиях активного развития инструментов–ассистентов в тестировании, основанных на AI, эта информация также помогла бы повысить точность формирования запросов к большой языковой модели и увеличила бы вероятность получения качественных рекомендаций по устранению найденных ODFlaky тестов.

Одним из таких инструментов является Explyt Test[5], разрабатываемый компанией Explyt в Санкт–Петербурге. Разработка инструмента началась в начале 2024 года, и уже к концу того же года состоялся его полноценный релиз, сопровождаемый презентациями на крупных конференциях Joker[6] [9] и HeisenBug[7] [10]. Ассистент совмещает в себе последние разработки в области AI с формальными методами, проводя анализ пользовательского кода перед формированием запросов в большую языковую модель. И поскольку Explyt Test позиционирует себя как инструмент всестороннего ухода за тестами, добавление модуля анализатора ODFlaky тестов стало бы отличным дополнением к уже существующим возможностям инструмента.

[5]Explyt Test: AI test generation. – URL: https://explyt.com
[6]Joker: Java conference for experienced developers. – URL: https://jokerconf.com
[7]HeisenBug: Conference on testing not just for testers – URL: https://heisenbug.ru

2. Постановка задачи

Целью данной работы является разработка основанного на AI подхода к анализу и устранению моргающих тестов, зависящих от порядка запуска, в инструменте Explyt Test.

Для достижения цели были выделены следующие задачи:

1. Исследовать предлагаемые подходы к анализу и устранению моргающих тестов, зависящих от порядка запуска, в научных публикациях и существующих инструментах
2. Разработать механизм выявления моргающего теста и приводящего его к морганию теста загрязнителя
3. Локализовать, если это возможно, инструкции в тесте загрязнителе, исполнение которых приводит к появлению моргающего теста
4. Сформировать запрос к LLM на основе полученной в процессе анализа тестов информации
5. Разработать способ представления данных, полученных в процессе анализа тестов, а также рекомендаций, предложенных LLM
6. Апробировать предложенное решение на наборе реальных проектов с моргающими тестами, зависимыми от порядка запуска

3. Обзор

3.1. Существующие решения

На сегодняшний день проведен уже целый ряд исследований на тему выявления и устранения ODFlaky тестов. Обзорная статья [2], систематизирующая существующие подходы и достижения в этой области, стала основным источником поиска возможных методов решения поставленных задач.

Всю работу по выявлению и устранению ODFlaky тестов можно разделить на 3 части: поиск порядкозависимых тестов, анализ порядкозависимых тестов и их устранение. И если поиск таких тестов во всех исследованиях производится примерно по одной схеме: запуск тестов в разном порядке, то анализ и устранение имеют уже более разнообразные подходы.

3.1.1. Анализ ODFlaky тестов

PolDet [8] — инструмент для обнаружения тестов, выполнение которых изменяет общее состояние системы и тем самым создает зависимости между тестами. Конечно, при использовании формальных методов для дальнейшего устранения ODFlaky тестов, информация обо всех изменяющихся во время выполнения теста состояниях ценна. Однако в контексте применения LLM этот подход может оказаться менее эффективным, при всей сложности его реализации, поскольку он не предоставляет конкретных подсказок для устранения ODFlaky тестов.

FlakyDoctor [5] для каждого ODFlaky теста находит все тесты, выполнение которых приводит к его морганию, а затем пытается локализовать причины возникновения моргания, анализируя stack trace запуска тестов для нахождения не прошедшего assert'а, собирая информацию обо всех статических полях тестового класса и методах для обновления состояний. Конечно, такая информация может быть полезной при формировании запроса к LLM, однако и она не предоставляет конкретных подсказок по устранению ODFlaky тестов.

3.1.2. Устранение ODFlaky тестов

Существует 3 известных инструмента для устранения ODFlaky тестов: iFixFlakies [6] и ODRepair [7], построенные на применении формальных методов, а также FlakyDoctor, основанный на AI.

Согласно сравнению [5], проведенному разработчиками FlakyDoctor, их инструмент демонстрирует более высокую эффективность в устранении ODFlaky тестов, чем решения, основанные на формальных методах. На наборе из 332 ODFlaky тестов, FlakyDoctor смог исправить на 12% тестов больше, чем ODRepair, и на 17% больше, чем iFixFlakies.

Однако, можно предположить, что эффективность FlakyDoctor может стать еще выше, при улучшении проводимого анализа ODFlaky тестов. В частности, инструмент не учитывает, что LLM было бы полезно знать о конкретных инструкциях, исполнение которых привело к появлению ODFlaky теста. Также FlakyDoctor, при формировании запроса к LLM, не учитывает важность добавления реализаций методов, вызываемых в тестах, но находящихся за пределами тестового класса.

3.2. Технологии, используемые для разработки

Выбор стека технологий был обусловлен интеграцией решения в продукт Explyt Test.

Языки реализации: Kotlin[8], Java[9].

В процессе разработки были использованы основные механизмы IntelliJ Platform[10].

- **Actions**[11]. Механизм создания пользовательских команд в IntelliJ IDEA. Используется для добавления точки входа анализатора ODFlaky тестов.
- **Psi**[12]. Одна из ключевых частей архитектуры IntelliJ Platform, предоставляющая инструменты для работы с синтаксическим деревом проекта. Используется в реализации модуля анализатора ODFlaky тестов для работы с пользовательским кодом и взаимодействия с другими модулями Explyt Test.
- **User Interface Components**[13]. Механизм создания графического интерфейса в IntelliJ IDEA. Используется для представления результатов работы анализатора ODFlaky тестов.

Для модификации Explyt Test runtime модуля используются следующие технологии:

- **ASM**[14]. Фреймворк для манипуляций с байткодом. При реализации используется для конвертации массива байткода в абстрации ASM

[8]The Kotlin Programming Language. — URL: https://kotlinlang.org

[9]The Java Programming Language. – URL: https://www.java.com

[10]IntelliJ Platform. — URL: https://plugins.jetbrains.com/docs/intellij/welcome.html

[11]IntelliJ Platform, Actions. – URL: https://plugins.jetbrains.com/docs/intellij/basic-action-system.html

[12]IntelliJ Platform, Psi. – URL: https://plugins.jetbrains.com/docs/intellij/psi.html

[13]IntelliJ Platform, User Interface Components. – URL: https://plugins.jetbrains.com/docs/intellij/user-interface-components.html

[14]Java bytecode manipulation and analysis framework. – URL: https://asm.ow2.io

и обратно. Конвертация в абстракции ASM удобна для дальнейшей работы с библиотекой JacoDB.

- **JacoDB**[15]. Библиотека для работы с байткодом. При реализации используется для анализа и дальнейшей трансформации тестовых методов.

Для взаимодействия с LLM на стороне Explyt Test используется Explyt AI-client. Формирование запроса происходит с использованием одной из реализаций интерфейса `LLMRequest`.

[15]JacoDB. Fast and effective way to access and analyze java bytecode. – URL: https://github.com/UnitTestBot/jacodb

Заключение

В ходе выполнения работы были решены следующие задачи:

1. Исследованы предлагаемые подходы к анализу и устранению моргающих тестов, зависящих от порядка запуска, в научных публикациях и существующих инструментах
2. Разработан механизм выявления моргающего теста и приводящего его к морганию теста загрязнителя
3. Частично разработан механизм локализации инструкций в тесте загрязнителе, исполнение которых приводит к появлению моргающего теста
4. Реализован прототип полнофункционального решения
5. Проведена апробация анализатора на модельных примерах

Планируется решить следующие задачи:

1. Доработать механизм локализации инструкций в тесте загрязнителе, исполнение которых приводит к появлению моргающего теста
2. Сформировать запрос к LLM на основе полученной в процессе анализа тестов информации
3. Разработать способ представления данных, полученных в процессе анализа тестов, а также рекомендаций, предложенных LLM
4. Апробировать предложенное решение на наборе реальных проектов с моргающими тестами, зависимыми от порядка запуска

Код анализатора не может быть предоставлен, поскольку находится под договором о неразглашении.

Список литературы

[1] A. Tahir, S. Rasheed, J. Dietrich, N. Hashemi, and Lu Zhang, *"Test flakiness' causes, detection, impact and responses: A multivocal review," Journal of Systems and Software, Volume 206*, 2023, doi: https://doi.org/10.1016/j.jss.2023.111837 (дата обращения: 8 января 2025 г.)

[2] O. Parry, G. M. Kapfhammer, M. Hilton, and P. McMinn, *"A Survey of Flaky Tests," ACM Trans. Softw. Eng. Methodol. 31, 1, Article 17*, 2022, 74 pages, doi: https://doi.org/10.1145/3476105 (дата обращения: 8 января 2025 г.)

[3] P. Zhang, Y. Jiang, A. Wei, V. Stodden, D. Marinov, and A. Shi, *"Domain-Specific Fixes for Flaky Tests with Wrong Assumptions on Underdetermined Specifications," In Proceedings of the 43rd International Conference on Software Engineering* (ICSE '21), IEEE Press, pp. 50–61, doi: https://doi.org/10.1109/ICSE43902.2021.00018 (дата обращения: 8 января 2025 г.)

[4] Y. Pei, J. Sohn, S. Habchi, and M. Papadakis, *"Non-Flaky and Nearly-Optimal Time-based Treatment of Asynchronous Wait Web Tests," ACM Trans. Softw. Eng. Methodol. Just Accepted*, 2024, doi: https://doi.org/10.1145/3695989 (дата обращения: 8 января 2025 г.)

[5] Y. Chen and R. Jabbarvand, *"2024. Neurosymbolic Repair of Test Flakiness," In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (ISSTA 2024), Association for Computing Machinery, New York, NY, USA, pp. 1402–1414, doi: https://doi.org/10.1145/3650212.3680369 (дата обращения: 8 января 2025 г.)

[6] A. Shi, W. Lam, R. Oei, T. Xie, and D. Marinov, *"IFixFlakies: a framework for automatically fixing order-dependent flaky tests," In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software*

Engineering (ESEC/FSE 2019), Association for Computing Machinery, New York, NY, USA, pp. 545–555, doi: https://doi.org/10.1145/3338906.3338925 (дата обращения: 8 января 2025 г.)

[7] C. Li, C. Zhu, W. Wang, and A. Shi, *"Repairing order-dependent flaky tests via test generation," In Proceedings of the 44th International Conference on Software Engineering* (ICSE '22), Association for Computing Machinery, New York, NY, USA, pp. 1881–1892, doi: https://doi.org/10.1145/3510003.3510173 (дата обращения: 8 января 2025 г.)

[8] A. Gyori, A. Shi, F. Hariri, and D. Marinov, *"Reliable testing: detecting state-polluting tests to prevent test dependency," In Proceedings of the 2015 International Symposium on Software Testing and Analysis* (ISSTA 2015), Association for Computing Machinery, New York, NY, USA, pp. 23–233, doi: https://doi.org/10.1145/2771783.2771793 (дата обращения: 8 января 2025 г.)

[9] E. Kulikov and I. Muravev, *"Automatic Test Generation is Like Fighting Dragons,"* url: https://jokerconf.com/en/talks/5b4dd75f9453450393121e835fc09eb8/ (дата обращения: 8 января 2025 г.)

[10] E. Kulikov and I. Muravev, *"Невыносимая легкость" автоматической генерации тестов,"* url: https://heisenbug.ru/archive/2024%20Autumn/talks/aa0de44a3294492b9e9a17eb1cdf0fa4/ (дата обращения: 8 января 2025 г.)