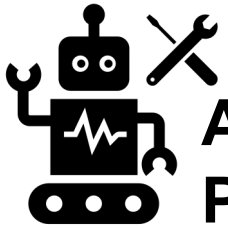
JUnit +  Automated Program Repair

RUB

RUHR-UNIVERSITÄT BOCHUM

JUnit meets APR

Development of a JUnit Extension for Automated Program Repair

Motivation and Background

- Automated program repair (APR)[1] promises great help for software developers by automatically generating patches. Various techniques for APR (search-based, semantic-based, template-based, ML-based) have been proposed in the research community, but only a few have made their way to the application in practice.
- This project aims to provide a concrete solution that integrates with existing workflows like JUnit[2]. JUnit is the standard unit testing framework for Java. Failing test cases usually indicate a regression error, meaning that an error was introduced with recent changes.
- An automated repair solution for JUnit would follow up on the failing test case event and automatically propose a patch for the developer. Therefore, it attempts to merge testing and repair, which is an essential step to improve software quality at scale.

[1] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. 2019. Automated program repair. Commun. ACM 62, 12 (December 2019), 56–65. https://doi.org/10.1145/3318162

[2] https://junit.org/junit5/

Student Task and Responsibilities

- IDE (VSCode or IntelliJ) extension that allows the developer to run tests with JUnit followed by the automated repair of the failing tests by adapting the corresponding Java code. This represents the complete workflow from bug detection, fault localization, fix generation, and patch application/integration.
- The students are supposed to explore existing APR techniques and should also implement their own variant/new technique. A starting point can be existing APR libraries[3].
- The implementation(s) must be evaluated on open-source projects, which can be selected by the students. The findings of the conducted experiments must be thoroughly documented.

[3] Matias Martinez and Martin Monperrus. ASTOR: a program repair library for Java (demo). ISSTA 2016. https://doi.org/10.1145/2931037.2948705

Minimum Expectation and Extensions

- The minimum expected result is the implementation of an IDE extension that supports the described automated functionality generating and integrating a patch for a failing JUnit test case. The group must integrate/implement at least one APR approach.
- An important aspect of this project is the design of the implementation, the maintainability, and the extensibility. Further, the implementation should be properly tested.
- The group must show the operationability of their implementation with a live demonstration.
- The usage of Large Language Models (LLMs) in an LLM-based APR component is an option.
- Potential extension points:
  - The group can implement multiple APR variants and compare their efficacy in different scenarios.
  - The group can explore various interaction possibilities (e.g., interactive features to present the generated fault locations or patches), which will help the developer to understand the problem with the failing test case, select the correct patch, or formulate their own patch.

Initial Timeplan

Week 1 (07.10 – 13.10.)[M1]	Kick-Off & Introduction
Week 2 (14.10 – 20.10.)	Planning, Requirements Engineering, and Design
Week 3 (21.10 – 27.10.)[M2]	
Week 4 (28.10. – 03.11.)	***1st Coding Cycle*** Implementation of prototype (no complete workflow implementation expected)
Week 5 (04.11. – 10.11.)	
Week 6 (11.11. – 17.11.)	
Week 7 (18.11. – 24.11.)[M3]	
Week 8 (25.11. – 01.12.)	***2nd Coding Cycle*** Implementation of complete workflow
Week 9 (02.12. – 08.12.)	
Week 10 (09.12. – 15.12.)[M4]	
Week 11 (16.12. – 22.12.)	***3rd Coding Cycle*** Improvements and extensions
Week 12 (23.12. – 29.12.)*	
Week 13 (30.12. – 05.01.)*	
Week 14 (06.01. – 12.12.)	
Week 15 (13.01. – 19.01.)	Finalization code (code freeze after week 16)
Week 16 (20.01. – 26.01.)[M5]	
Week 17 (27.01. – 02.02.)	Final documentation and report writing/submission
Week 18 (03.02. – 07.02.)[M6]	

** Christmas Holidays*

Milestones:

- M1: Project Kick-Off
- M2: Code Design Submission
- M3: Demonstration of Prototype
- M4: Demonstration of Complete Workflow
- M5: Final Code Submission
- M6: Final Report Submission

Working Mode

- Weekly meetings with advisor (will be arranged taking into account all schedules)
- Expected are at least one additional weekly group-internal meeting and active discussions on Slack/Discord
- Kick-Off Meeting: Monday, 7th October 2024 (details will be announced)

Other Information

- **Prerequisites:** Programming experience, preferably in Java, is needed for this project. Further, it would be beneficial to have experience with JUnit Testing and IDE plugin development. Prior knowledge in automated program repair is not needed.
- **Deliverables:** Source code, its documentation, and a publishable report (incl. the evaluation results), ideally to submit to a conference (e.g., as a Demo paper) or at least publish as a technical report on arxiv.org.
- **Number of Participants:** 2-4
- **Target Group:** Bachelor and Master students
- **Industrial partner:** None (done at RUB)

Contact

Prof. Dr. Yannic Noller

- Raum: MC 4.114
- E-Mail: yannic.noller@rub.de
- Office hours: By Arrangement

- https://informatik.rub.de/ac-personen/yannic-noller/
- https://yannicnoller.github.io