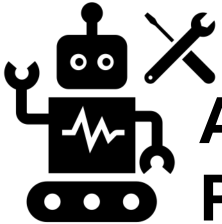
JUnit +



Automated Program Repair

RUB

RUHR-UNIVERSITÄT BOCHUM

JUnit meets APR

Development of a JUnit Extension for Automated Program Repair

Prof. Dr. Yannic Noller
Software Quality group

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

- Realize an 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.
- A previous study project already provided the initial implementation of such extension. This new study project is supposed to build on top of it and add new features.
- The students are supposed to explore existing LLM-based APR techniques and should also implement their own variant/new technique. A starting point can be works like ChatRepair[3] and AutoCodeRover[4].
- 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] C. S. Xia and L. Zhang, „Automated Program Repair via Conversation: Fixing 162 out of 337 Bugs for $0.42 Each using ChatGPT“, ISSTA 2024. https://doi.org/10.1145/3650212.3680323

[4] Y. Zhang, H. Ruan, Z. Fan, and A. Roychoudhury, „AutoCodeRover: Autonomous Program Improvement“, ISSTA 2024. https://doi.org/10.1145/3650212.3680384

Minimum Expectation and Extensions

- The minimum expected result is the implementation of an IDE extension that supports the described functionality generating and integrating a patch for a failing JUnit test case.
- The group must implement at least one additional (not yet integrated) APR approach and integrate the functionality to combine patches from two APR components. Further, the group should integrate/implement at least one LLM-based APR component. The set of desired features will be discussed during the first project phase.
- 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.
- Potential extension points:
  - Explore more interactive features to present the generated fault locations or patches, which will help the developer to understand and fix the problem.
  - Explore options to repair syntax errors, i.e., non-compiling code.

Initial Timeplan

Week 1 (13.10. – 19.10.) [M1]	Kick-Off & Introduction
Week 2 (20.10. – 26.10.)	Planning, Requirements Engineering, and Design
Week 3 (27.10. – 02.11.) [M2]	
Week 4 (03.11. – 09.11.)	***1st Coding Cycle*** Implementation of prototype (no complete workflow implementation expected)
Week 5 (10.11. – 16.11.)	
Week 6 (17.11. – 23.11.)	
Week 7 (24.11. – 30.11.) [M3]	
Week 8 (01.12. – 07.12.)	***2nd Coding Cycle*** Implementation of complete workflow
Week 9 (08.12. – 14.12.)	
Week 10 (15.12. – 21.12.) [M4]	

Week 11 (22.12. – 28.12.)*	***3rd Coding Cycle*** Improvements and extensions
Week 12 (29.12.. – 04.01.)*	
Week 13 (05.01. – 11.01.)	
Week 14 (12.01. – 18.01.)	
Week 15 (19.01. – 25.01.) [M5]	Finalization code (code freeze after week 15)
Week 16 (26.01. – 01.02.)	Final documentation and report writing/submission
Week 17 (02.02. – 08.02.) [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: in the week of 13th October 2025 (details will be announced)

Other Information

- **Prerequisites:** Programming experience, preferably in Java, is needed for this project. Further, having experience with JUnit Testing and IDE plugin development would be beneficial. Prior knowledge of automated program repair is not needed. Prior attendance of the Software Engineering course is highly recommended.
- **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