FRAFOL: FRAmework FOr Learning mutation testing

Pedro Tavares
Faculty of Engineering, University of Porto
Porto, Portugal
up201406991@edu.fe.up.pt

Ana Paiva
INESC TEC, Faculty of Engineering, University of Porto
Porto, Portugal
apaiva@fe.up.pt

Domenico Amalfitano
University of Naples "Federico II"
Naples, Italy
domenico.amalfitano@unina.it

René Just
University of Washington
Seattle, USA
rjust@cs.washington.edu

Abstract

Mutation testing has evolved beyond academic research, is deployed in industrial and open-source settings, and is increasingly part of universities' software engineering curricula. While many mutation testing tools exist, each with different strengths and weaknesses, integrating them into educational activities and exercises remains challenging due to the tools' complexity and the need to integrate them into a development environment. Additionally, it may be desirable to use different tools so that students can explore differences, e.g., in the types or numbers of generated mutants. Asking students to install and learn multiple tools would only compound technical complexity and likely result in unwanted differences in how and what students learn.

This paper presents FRAFOL, a framework for learning mutation testing. FRAFOL provides a common environment for using different mutation testing tools in an educational setting.

CCS Concepts

• **Software and its engineering** → **Software testing and debugging**; • **Social and professional topics** → **Software engineering education**.

Keywords

Software Testing, Mutation Testing, Teaching Mutation Testing, Teaching Tool

ACM Reference Format:
Pedro Tavares, Ana Paiva, Domenico Amalfitano, and René Just. 2024. FRAFOL: FRAmework FOr Learning mutation testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3650212.3685306

1 Introduction

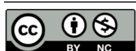
Mutation testing has evolved beyond academic research and is now used in various open-source and industrial settings (e.g., [3, 16]). The increasing complexity of software systems and the need for robust testing methodologies have made mutation testing an essential skill for software engineers [1].

Mutation analysis involves creating faulty program variants (mutants) using well-defined rules (mutation operators) on syntactic descriptions to systematically alter syntax or related objects [14]. This technique has been effectively employed in research to evaluate test efficacy and aid in testing and debugging. Given an original program and a corresponding test suite, mutation analysis creates a set of mutants for the original program, and the mutant-detection ratio of the test suite indicates its effectiveness. Empirical studies support using mutants as proxies for real faults [2, 4, 13]. Another application of mutation analysis is automated debugging, where mutants help locate faults or iteratively modify a program until it meets a specification, such as passing all tests in a test suite [9, 10].

Mutation testing builds on top of mutation analysis and uses undetected mutants as test goals to enhance a test suite. Mutation testing was once considered impractical due to the large number of mutants that can be generated, even for small programs. However, it is increasingly adopted in the industry thanks to new approaches like incremental, commit-level mutation, suppression of unproductive mutants, and focusing on individual mutants rather than overall mutant detection ratios [3, 15, 17].

Teaching mutation testing is crucial to prepare the next generation of software engineers to effectively employ this technique and improve software quality. As mutation testing becomes more and more prevalent in industry, there is a need to educate software engineers about this technique, and applications of program mutations more generally. However, teaching mutation testing presents several challenges [8]. Software testing can often be very complex and requires a deep understanding of the codebase and its principles. In practice, mutation testing can be resource-intensive and require a considerable amount of effort to setup, configure, and integrate it into an existing development workflow. Integrating mutation testing into an existing software engineering curriculum is challenging, as it requires balancing learning goals and theoretical concepts with technical complexity and realistic applications.

In software engineering education, the scientific community has grown interested in developing tools that strike the right balance between usability and realism and enable students to learn software testing techniques effectively. Two examples for innovative tools are CodeDefenders [7] and Web-CAT [5]. CodeDefenders introduced a game-based learning approach. Students play as "Attackers", inserting defects into a class under test, or "Defenders", writing tests to detect and guard against these defects. This interactive setup boosts



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

ISSTA '24, September 16–20, 2024, Vienna, Austria
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0612-7/24/09
https://doi.org/10.1145/3650212.3685306

engagement and sharpens test-writing skills, improving student performance. Web-CAT (Web-based Center for Automated Testing) automates grading and emphasizes quality in student-written test cases. It promotes an understanding of expected code behaviour through test-driven development principles. Web-CAT's detailed feedback enhances learning outcomes and reduces grading workload, making it a valuable tool in software testing education. CodeDefenders and Web-CAT aim to provide students with practical, engaging, and effective learning experiences.

Many software testing tools are readily available, but they often lack comprehensive features and consistency in integration into curricula, necessitating multiple setup procedures and extensive prior knowledge. In the context of mutation testing for Java, numerous tools have been proposed in the literature, each with its unique features and capabilities, but the primary focus of these tools is applications in practice or research [1]. There is a shortage of tools that enable students to quickly gain hands-on experience with various mutation testing tools and start writing tests.

This paper proposes FRAFOL, a "FRAmework FOr Learning mutation testing". FRAFOL aims to simplify the setup and configuration requirements by offering a uniform environment for learning, and experimenting with, mutation testing.

2 The FRAFOL tool

This section explains FRAFOL's design rationale and provides implementation details. FRAFOL provides a common environment through which a student can use two different mutation tools, Major and PIT. We chose these two mutation testing tools because they are widely used in research and practice and they have complementary features (e.g., source code vs. bytecode mutations). FRAFOL is not specific to these two tools, and others could be integrated. FRAFOL shields students from technical complexity but nonetheless allows them to observe key differences, such as types of mutants and how difficult it is to detect them. Furthermore, we implemented FRAFOL as an extension of Defects4J [12], which is a collection of reproducible Java bugs with an infrastructure of tools and scripts and which is widely used in research and education. We chose Defects4J because it provides uniform access to numerous realistic subjects, including compiling and testing them.

FRAFOL aims to provide an easy-to-install and easy-to-use environment where students can immediately start analyzing mutants and writing tests to detect them without spending time figuring out the intricacies of the mutation testing tools.

FRAFOL provides the following eight main features:

- F1: Provides a web interface.
- F2: Allows selecting a project version from the Defects4J.
- F3: Allows selecting a mutation tool to work with (Major [11] or PIT [6]).
- F4: Provides an IDE to i) analyze the Java code of the selected project version and ii) develop a `JUnit` test class.
- F5: Allows compiling the `JUnit` test class.
- F6: Supports executing the selected mutation tool and the analysis of the results.
- F7: Evaluates and shows the following metrics: the number of generated mutants, the number of killed mutants by the `JUnit` test class, the number of live mutants, and the code coverage achieved by the `JUnit` test class in terms of LoC coverage and branch coverage.
- F8: Shows details about live mutants to aid students in their mutation testing effort. These details include an ID, the mutated source code's line number, the mutated method's name, and the applied mutation operator.

2.1 Design and Implementation

FRAFOL uses the Defects4J benchmark as its core component. Defects4J already interfaces with the Major mutation testing tool, and it provides automation for various analyses, including mutation analysis and code coverage. Additionally, Defects4J provides uniform access to a repository of Java projects, each with corresponding test cases produced by developers. However, Defects4J is command-line driven and integration into a development environment (e.g., for mutation testing) is left to end users. Figure 1 shows a UML component diagram of the FRAFOL tool: FRAFOL extends Defects4J by integrating the PIT mutation tool and providing an Adapter to expose a unified Common Mutation API for both mutation tools. The adapter provides a unified interface, allowing

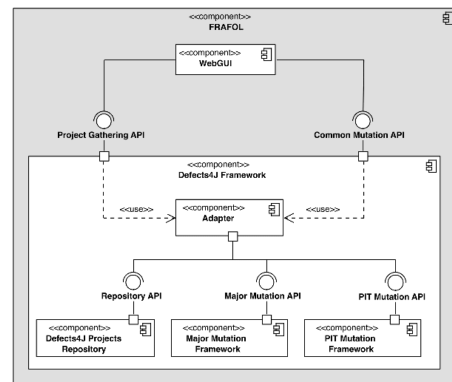


Figure 1: UML Component Diagram of the FRAFOL tool

different mutation testing tools to be executed using the same commands. The Adapter also exposes a Project Gathering API providing features for querying the repository of Defects4J projects. FRAFOL also provides a WebGUI component, implemented in Python using Flask. We designed the GUI to provide a rich and interactive web interface that simplifies the use of the Defects4J Framework APIs and offers an integrated development environment. We chose this approach because previous work reported that students and instructors prefer such a setup for educational purposes [1].

FRAFOL is integrated into a Docker-based, containerized architecture to enhance end-user experience by simplifying deployment and ensuring consistency and efficient dependency management. Figure 2 shows the Docker-based container architecture implemented, allowing FRAFOL to operate in a controlled environment. As illustrated, Docker implements a client-server architecture, where the client communicates with the Docker daemon to build and run containers. The container executes FRAFOL on an instance of Ubuntu. The WebGUI manages interaction with FRAFOL, acting as the communication bridge between the client and the GUI within the container.

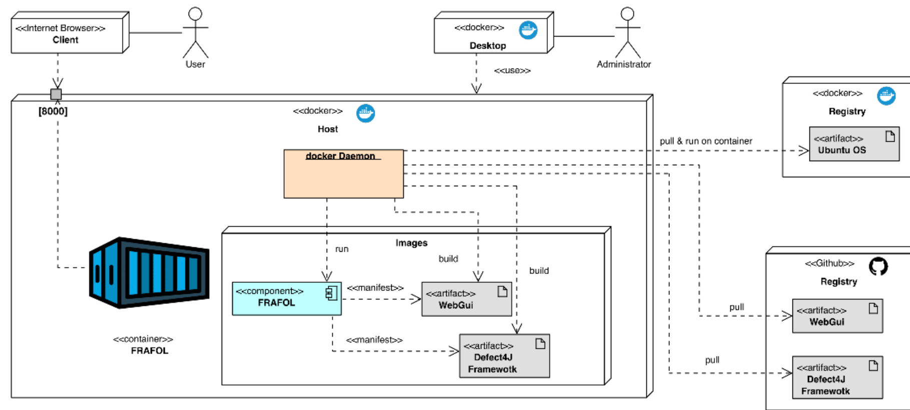


Figure 2: UML Deployment Diagram of the FRAFOL tool

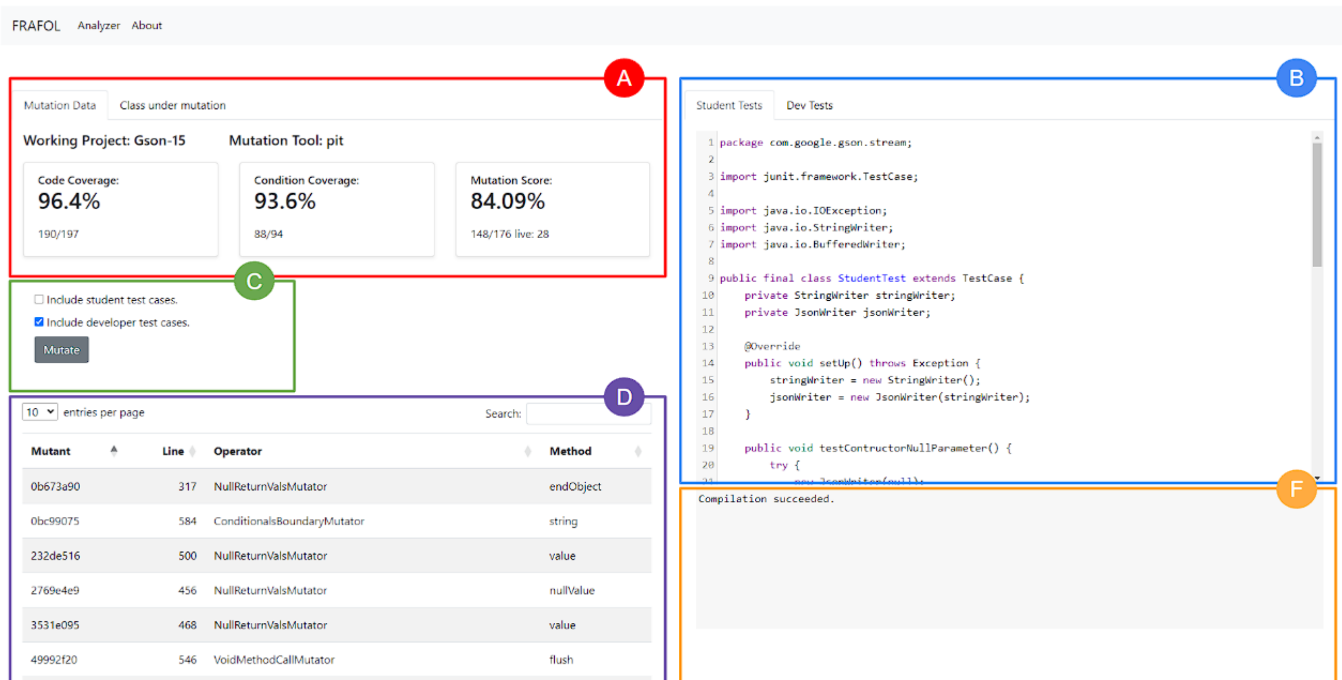


Figure 3: FRAFOL Mutation analysis integrated environment

2.2 User Interface

In the current configuration, where the Docker is installed in the users' local machine, FRAFOL can be accessed through the browser at http://localhost:8000/.

The first webpage of FRAFOL shows a list of Java projects available in Defects4J and their corresponding versions. A user starts by importing a desired project version, which results in FRAFOL obtaining and compiling the corresponding source code, and adding the version to the list of available project versions. Afterward, a user can `Open` an imported project version and also select a mutation testing tool to work with.

FRAFOL then configures an environment for the user selection and redirects the user to FRAFOL's primary web interface (Figure 3).

Figure 3 in (**A**) presents two tabs: one with a dashboard featuring infographics on code coverage, condition coverage, and mutation score data, and another with the code of the class under mutation. (**B**) also includes two tabs, one displaying the source code for the class under mutation and the other showing existing developer-written test cases.

The user can execute the mutation tool by clicking the `Mutate` button (**C**). FRAFOL launches the mutation tool and runs the test suite against the generated mutants. The user may opt to run only existing developer tests, newly developed student tests, or both. The compilation results for "Student Tests" class may be

seen at (**F**). FRAFOL presents a table containing data on all remaining live mutants (**D**). This data comprises `mutantId`, `codeLine`, `mutantOperator`, and `classMethod`. Given the provided information, the user can then analyze any live mutants, inspect the source code that was mutated, and write test cases that detect live mutants. If a test case detects a previously live mutant, such mutant is removed from the table.

2.3 Tool Availability

FRAFOL is available at https://github.com/projFRAFOL/projFRAFOL (see README.md for details), and a video demonstration is available at: https://youtu.be/JMvGskRQre8. The initial version of FRAFOL supports the Gson-15 and Cli-32 project versions. We are currently working on a generalization that will allow other Defects4J project versions to be included.

3 Validation

We conducted two user studies—a formative study that informed the design of FRAFOL, and a summative study that evaluated the current version of the tool. The formative study, involving 35 students, using an early prototype, revealed a need for the following key improvements: (1) easy installation through Dockerization, (2) streamlined and unified GUI, (3) editing capabilities for test cases in the same web environment, and (4) a side-by-side view of code and tests. The summative evaluation asked four MSc students to evaluate various aspects of FRAFOL, such as its usability and complexity (see Table 1). These students had previously completed a software testing course where they learned about mutation testing. The study was carried out remotely and involved three steps:

First, the students installed FRAFOL, following the provided guide, which took approximately 15 minutes. Second, one of the authors gave a 10-minute presentation on FRAFOL. Third, the students were given a list of four tasks to complete within 35 minutes:

(1) Select the Gson project, specifically version 15, and choose the PIT mutation testing tool.
(2) Analyze only the existing developer tests and assess their code coverage and mutation score (**C** in Figure 3).
(3) Analyze the results (**D** in Figure 3) and write a `JUnit test class` (**B** in Figure 3) to target a specific mutant form the list of the live mutants (**D** in Figure 3).
(4) Reevaluate the mutation coverage and score, this time including developer and student test cases (**C** in Figure 3).

Finally, the students completed a questionnaire[1], which consisted of closed questions with responses based on a 5-level Likert scale, ranging from 1 (strongly disagree) to 5 (strongly agree). Table 1 shows the aggregated results of the questionnaire.

The results indicate that students highly appreciated the `Interface` (4.3) and expressed a strong sense of `Satisfaction` while using the tool (4.2). They believed that FRAFOL could significantly enhance their learning `Effectiveness` (4.1) and found it **`Useful` (4.0)**. The `Usability` and `Learnability` of FRAFOL both received a score of 3.9. Although these are positive results, there is room for improvement (e.g., by providing a short video tutorial). The `System feedback` received a score of 3.4. Since the user study, we have implemented a new mechanism to assist students further: by selecting a live mutant in the summary table, students are redirected to the specific line of source code that was mutated. The perceived `Complexity` was rated at 1.4. As a negatively phrased question, the low score means that students do not find the tool complex or difficult to understand, which is a positive outcome.

[1] https://forms.gle/V2C3tqaZWK4YsYkK9

Table 1: Students' assessment of FRAFOL

Usability	3.9
Learnability	3.9
Satisfaction	4.2
Complexity	1.4
Effectiveness	4.1
Usefulness	4.0
System feedback	3.4
Interface	4.3

4 Conclusions and Future Work

This paper presented FRAFOL (Framework For Learning Mutation Testing). FRAFOL builds on top of the Defects4J framework, integrates the PIT and Major mutation testing tools, and offers a unified interface for mutation testing. The tool's WebGUI and Docker-based deployment streamline usability and accessibility, making it ideal for teaching and learning mutation testing. FRAFOL aims at providing a user-friendly environment that supports educators and students in mastering mutation testing techniques. Moreover, the preliminary experimentation with four students provided valuable insights into the usability and effectiveness of FRAFOL, setting a promising foundation for further refinement and expansion of the tool in educational settings.

Future work includes enhancing FRAFOL with advanced features to aid mutation testing, such as displaying code coverage metrics, visualizing control flow graphs of the code under test, and conducting analyses on live and productive mutants. Additionally, we aim to streamline accessibility by enabling installation on remote servers, eliminating the need for local Docker installations. Furthermore, plans involve expanding experimentation to a broader student base and integrating FRAFOL into courses tailored for industrial professionals, thereby enhancing its practical utility and educational impact. Finally, we plan to incorporate a dedicated set of GUIs within FRAFOL designed for teachers. This will make it easier to manage and monitor students' progress, provide comprehensive analytics on their performance, and offer intuitive interfaces for creating and administering mutation testing tasks. Lastly, we intend to integrate additional mutation tools in FRAFOL.

Acknowledgments

This work is being funded by the ENACTEST Erasmus+ project number 101055874. René Just's work is supported in part by National Science Foundation grants CCF-1942055 and CNS-2120070.

References

[1] Domenico Amalfitano, Ana C. R. Paiva, Alexis Inquel, Luís Pinto, Anna Rita Fasolino, and René Just. 2022. How do Java mutation tools differ? *Commun. ACM* 65, 12 (nov 2022), 74–89. https://doi.org/10.1145/3526099

[2] James H Andrews, Lionel C Briand, and Yvan Labiche. 2005. Is mutation an appropriate tool for testing experiments?. In *Proceedings of the 27th international conference on Software engineering*. 402–411.

[3] Moritz Beller, Chu-Pan Wong, Johannes Bader, Andrew Scott, Mateusz Machalica, Satish Chandra, and Erik Meijer. 2021. What It Would Take to Use Mutation Testing in Industry—A Study at Facebook. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 268–277. https://doi.org/10.1109/ICSE-SEIP52600.2021.00036

[4] Yiqun T Chen, Rahul Gopinath, Anita Tadakamalla, Michael D Ernst, Reid Holmes, Gordon Fraser, Paul Ammann, and René Just. 2020. Revisiting the relationship between fault detection, test adequacy criteria, and test set size. In *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*. 237–249.

[5] P. J. Clarke, A. A. Allen, T. M. King, E. L. Jones, and P. Natesan. [n. d.]. Using a web-based repository to integrate testing tools into programming courses. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, SPLASH '10*. 193–200. https://doi.org/10.1145/1869542.1869573

[6] Henry Coles, Thomas Laurent, Christopher Henard, Mike Papadakis, and Anthony Ventresque. 2016. PIT: a practical mutation testing tool for Java (demo). In *Proceedings of the 25th International Symposium on Software Testing and Analysis* (Saarbrücken, Germany) *(ISSTA 2016)*. Association for Computing Machinery, New York, NY, USA, 449–452. https://doi.org/10.1145/2931037.2948707

[7] Gordon Fraser, Alessio Gambi, and José Miguel Rojas. 2020. Teaching Software Testing with the Code Defenders Testing Game: Experiences and Improvements. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 461–464. https://doi.org/10.1109/ICSTW50294.2020.00082

[8] V. Garousi, M. Felderer, M. Kuhrmann, K. Herkiloğlu, and S. Eldh. 2020. Exploring the industry's challenges in software testing: An empirical study. *Journal of Software: Evolution and Process* 32, 8 (2020). https://doi.org/10.1002/smr.2251

[9] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. 2019. Automatic Software Repair: A Survey. *IEEE Transactions on Software Engineering* 45, 1 (2019), 34–67. https://doi.org/10.1109/TSE.2017.2755013

[10] Ali Ghanbari, Samuel Benton, and Lingming Zhang. 2019. Practical program repair via bytecode mutation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Beijing, China) *(ISSTA 2019)*. Association for Computing Machinery, New York, NY, USA, 19–30. https://doi.org/10.1145/3293882.3330559

[11] René Just. 2014. The Major mutation framework: Efficient and scalable mutation analysis for Java. In *Proceedings of the 2014 international symposium on software testing and analysis*. 433–436.

[12] René Just, Darioush Jalali, and Michael D Ernst. 2014. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the 2014 international symposium on software testing and analysis*. 437–440.

[13] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 654–665. https://doi.org/10.1145/2635868.2635929

[14] A. Jefferson Offutt and Roland H. Untch. 2001. *Mutation 2000: Uniting the Orthogonal*. Springer US, Boston, MA, 34–44. https://doi.org/10.1007/978-1-4757-5939-6_7

[15] Goran Petrović, Marko Ivanković, Gordon Fraser, and René Just. 2021. Does mutation testing improve testing practices?. In *Proceedings of the 43rd International Conference on Software Engineering* (Madrid, Spain) *(ICSE '21)*. IEEE Press, 910–921. https://doi.org/10.1109/ICSE43902.2021.00087

[16] Goran Petrović, Marko Ivanković, Gordon Fraser, and René Just. 2021. Practical mutation testing at scale: A view from google. *IEEE Transactions on Software Engineering* 48, 10 (2021), 3900–3912.

[17] Goran Petrovic, Marko Ivankovic, Bob Kurtz, Paul Ammann, and René Just. 2018. An Industrial Application of Mutation Testing: Lessons, Challenges, and Research Directions. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 47–53. https://doi.org/10.1109/ICSTW.2018.00027

Received 2024-07-05; accepted 2024-07-26