Dynamic Testing of GUI Exercises in Headless Environments

Benjamin Schmitz
School of Computation, Information and Technology
Technical University of Munich
Munich, Germany
schmitz@tum.de

***Abstract*—Testing Graphical User Interface applications in non-graphical environments is challenging, especially for auto-graders in large-scale programming courses, where traditional approaches often depend on a graphical operating system. This paper presents a headless testing approach that simulates user interactions and verifies application state. A reference implementation demonstrates that interactive applications can be assessed reliably in headless settings. The approach ran in a large-scale introductory programming course, assessing 1,020 submissions without a graphical subsystem. A case study confirms the system meets key requirements and delivers reproducible, scalable, and pedagogically valuable feedback.**

***Index Terms*—GUI testing, automated grading, headless testing, JavaFX, TestFX, computer science education**

I. Introduction

The growing demand for computer scientists has resulted in programming courses enrolling over a thousand students [1]. Manual grading does not scale in such large classes, making the manual assessment of programming exercises impractical and creating the need for automated assessment systems [1]–[3]. Auto-graders commonly execute submissions inside Docker containers to ensure isolation and reproducibility. These containers typically operate in environments, which lack display hardware and graphical capabilities. As a result, native Graphical User Interface (GUI) applications cannot launch, causing test execution to fail. These environments are headless. This limitation poses a major challenge when using auto-graders for GUI-based exercises [2], [4].

Existing approaches include static analysis, using test doubles for graphical logic, or comparing screen images via computer vision. However, static methods cannot confirm interactive responsiveness, test doubles increase boilerplate code or limit usable GUI classes, and computer vision requires a virtual display setup with significant overhead while being sensitive to slight user interface changes [5]–[7].

Our work establishes requirements for enabling dynamic testing of GUI-based exercises in auto-graders. The functional requirements are:

- Execute GUI tests in a headless environment.
- Simulate clicks, cursor movement, and text input.
- Inspect the application state to verify expected GUI behavior.
- Provide detailed, educational feedback based on test outcomes.

In addition to these functional aspects, the system must meet the following quality requirements:

- Deterministic execution for fair assessment.
- Fast feedback for an effective learning experience [1].
- Straightforward test authoring for instructors supported by utility methods.

II. Concept

The system enables dynamic testing of GUI-based exercises in headless environments, overcoming the limitations of auto-graders that cannot execute native GUI applications without a graphical subsystem.

Its architecture integrates two mechanisms that decouple the testing process: a headless windowing system that launches applications without display hardware, and a simulation layer that replaces manual input with programmatic user interaction. Tests can also run locally with a graphical window, supporting flexible usage scenarios.

Figure 1 illustrates the system's structure. The `Student GUI Application` under test depends on a windowing abstraction provided by the `Window Service`, which is implemented either by the operating system (`OS`) or the `Headless` variant. The `Tests` interact with the application through the `User Interaction Simulator`, which emulates user behavior via a dedicated service interface. This design decouples tests from low-level details of the GUI framework implementation.

The architecture enables fully automated interaction and verification of the application state in both graphical and headless environments.

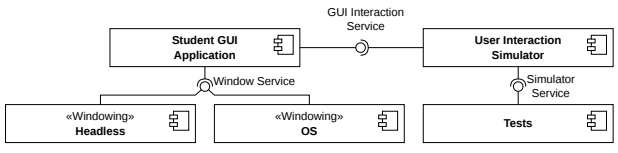

Fig. 1. Component Diagram showing the `Student GUI Application`, its dependency on alternative variants of the `Windowing` component and its relation to the `Tests` (UML Component Diagram)

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00393

III. Reference Implementation

A reference implementation, *Eos*[1], validates the proposed system. Eos targets the JavaFX[2] framework by using TestFX[3] to automate user interactions and verify the application state, and Monocle[4] to enable headless execution.

In this setup, students develop JavaFX-based GUI applications that integrate with Eos for test execution. Within the conceptual architecture shown in Figure 1, TestFX fulfills the role of the `User Interaction Simulator`. Monocle serves as the windowing implementation in environments without a conventional display manager, thus enabling headless execution.

When preparing an exercise with Eos, instructors provide both a template for the students and corresponding test cases. The template can range from a narrowly scoped skeleton, where only a few lines must be completed, to an open-ended starting point. The only fixed requirement is the entry point: the class and method used to launch the application must be predefined to ensure testability. Beyond this, students use plain JavaFX without needing to adapt their code for the framework.

Instructors write test cases with the standard TestFX API. These run headlessly in our framework without special adjustments. Eos extends TestFX with utility methods that simplify common tasks and provide richer feedback by default. These additions reduce boilerplate while staying compatible with TestFX.

Testing open-ended assignments remains more challenging. For instance, a start button might reasonably be labeled *start*, *launch*, or *execute*. In such cases, instructors can either prescribe naming conventions in the exercise description or write tests flexible enough to handle multiple correct variants. This challenge results from open-ended tasks rather than from Eos.

IV. Evaluation

We integrated the reference implementation into the auto-grading platform Artemis[5] and evaluated it in a case study in the course *Introduction to Programming*[6]. The course targets third-semester Management and Technology students and covers basic computer science, object-oriented programming, and GUI development.

We analyzed student submissions. Two exercises with predefined tests were deployed: a number conversion app (template-based, a few lines to complete) and a login form (open-ended design). The tutorial number conversion exercise involved 468 students (average score 78.5%) and was solved in guided tutorial sessions. The homework exercise involved 552 students (average score 95.3%) and was solved individually for a bonus grade. We clustered the feedback generated by the auto-grader with Levenshtein distance.

In the homework exercise, feedback pointed to isolated issues, such as missing labels (14 students) or incorrect image paths (30 students). In the tutorial exercise, frequent problems were wrong number conversions (53 students) or application crashes due to an intentional `null` placeholder in the template (42 students). This indicates a template issue to fix in future iterations.

To evaluate the quality requirements, we repeated test executions locally. Runtime scaled linearly with the number of test cases, and results remained consistent across runs unless the test itself introduced randomness.

These findings confirm determinism and suitability for real-time feedback in educational contexts [2], [5], [8]. The modular architecture, reusable exercise templates, and utility methods support instructors in creating headless GUI assignments.

The case study shows that dynamic headless testing generates actionable feedback for students while remaining reproducible and scalable.

V. Conclusion

This paper presents a concept for dynamic testing of GUI-based programming exercises in headless environments, addressing a key limitation of current auto-graders in computer science education. The modular architecture decouples execution from rendering and enables programmatic interaction.

To demonstrate the feasibility of this approach, we developed a reference implementation targeting JavaFX. By combining headless execution via Monocle with user interaction simulation and window state verification through TestFX, we created a reference implementation tailored to the needs of computer science education. We transferred and adapted these technologies to the educational domain, where automated testing and feedback are essential for scaling interactive assignments [1].

The case study shows that the system executes GUI tests reliably without graphical hardware, provides consistent feedback, and supports efficient test development. It processed 1,020 student submissions in a real-world course setting, fulfilling functional and quality requirements.

In the future, the reference implementation can be extended and evaluated on a larger scale, including surveys with students and instructors to gain deeper insights into usability, feedback quality, and learning impact. While the present work focuses on JavaFX, the concept is not limited to Java. Adaptations to other Java frameworks, such as Swing, or frameworks in other languages, such as PyQt6 with pytest-qt, may be possible. Such improvements broaden the range of supported GUI exercises and simplify large-scale grading.

Transparency in the use of AI tools

We used Grammarly (style), ChatGPT (text suggestions), Elicit (literature), and DeepL (translations). We carefully reviewed all AI-generated text.

[1] https://github.com/ls1intum/Eos
[2] https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm
[3] https://github.com/TestFX/TestFX
[4] https://wiki.openjdk.org/display/OpenJFX/Monocle
[5] https://github.com/ls1intum/Artemis
[6] https://aet.cit.tum.de/teaching/24w/itp/

REFERENCES

[1] S. Krusche, "Interactive learning - A scalable and adaptive learning approach for large courses," Habilitation, Technische Universität München, 2021.

[2] S. Krusche and A. Seitz, "ArTEMiS: An Automatic Assessment Management System for Interactive Learning," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, Feb. 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3159450.3159602

[3] T. Staubitz, H. Klement, J. Renz, R. Teusner, and C. Meinel, "Towards practical programming exercises and automated assessment in Massive Open Online Courses," in *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, 2015.

[4] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*, ser. The Addison-Wesley signature series. Addison-Wesley, 2013.

[5] A. Spillner and T. Linz, *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard*, 4th ed., ser. EBL-Schweitzer. dpunkt-Verl, 2010.

[6] T.-H. Chang, T. Yeh, and R. C. Miller, "GUI testing using computer vision," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2010. [Online]. Available: https://dl.acm.org/doi/10.1145/1753326.1753555

[7] U. Inoue, "GUI Testing for Introductory Object-Oriented Programming Exercises," in *Computational Science/Intelligence & Applied Informatics*, R. Lee, Ed. Springer International Publishing, 2019, vol. 787, series Title: Studies in Computational Intelligence. [Online]. Available: http://link.springer.com/10.1007/978-3-319-96806-3_1

[8] S. Krusche, A. Seitz, J. Börstler, and B. Bruegge, "Interactive Learning: Increasing Student Participation through Shorter Exercise Cycles," in *Proceedings of the Nineteenth Australasian Computing Education Conference*. ACM, Jan. 2017. [Online]. Available: https://dl.acm.org/doi/10.1145/3013499.3013513