UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS SOBRAL

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

MESTRADO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

LUCAS GABRIEL GUILHERME DOS SANTOS

CHALLENGES AND OPPORTUNITIES ON SOFTWARE TESTING TEACHING

SOBRAL

2024

LUCAS GABRIEL GUILHERME DOS SANTOS

CHALLENGES AND OPPORTUNITIES ON SOFTWARE TESTING TEACHING

Dissertação apresentada ao Curso de Mestrado em Engenharia Elétrica e de Computação do Programa de Pós-Graduação em Engenharia Elétrica e de Computação do Campus Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia Elétrica e de Computação. Área de Concentração: Engenharia Elétrica e de Computação

Orientador: Prof. Dr. Fischer Ferreira Jônatas

SOBRAL

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S1d SANTOS, LUCAS GABRIEL GUILHERME DOS.
Desafios e oportunidades no ensino de testes de software / LUCAS GABRIEL GUILHERME DOS SANTOS. – 2024.
66 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Campus de Sobral, Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Sobral, 2024.
Orientação: Prof. Dr. FISCHER JONATAS FERREIRA.

1. Software Testing. 2. Experiment. 3. EvoSuite. 4. ChatGPT. I. Título.

CDD 621.3

LUCAS GABRIEL GUILHERME DOS SANTOS

CHALLENGES AND OPPORTUNITIES ON SOFTWARE TESTING TEACHING

Dissertação apresentada ao Curso de Mestrado em Engenharia Elétrica e de Computação do Programa de Pós-Graduação em Engenharia Elétrica e de Computação do Campus Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia Elétrica e de Computação. Área de Concentração: Engenharia Elétrica e de Computação

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Fischer Ferreira Jônatas (Orientador)
Universidade Federal de Itajubá(UNIFEI)

Prof. Dr. Maurício Ronny de Almeida Souza
Universidade Federal de Lavras (UFLA)

Prof. Dr. Gustavo Henrique de Magalhães Gomes
Universidade Federal de Itajubá(UNIFEI)

Prof. Dr. Iális Cavalcante de Paula Júnior
Universidade Federal do Ceará (UFC)

Prof. Dr. Wendley Souza Silva
Universidade Federal do Ceará (UFC)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Aos meus amigos, por dividirem comigo os momentos alegres e desafiadores desta jornada, tornando-a mais leve e significativa. Aos meus professores, por compartilharem seus conhecimentos e me inspirarem a buscar a excelência.

ACKNOWLEDGEMENTS

Em primeiro lugar, agradeço a Deus pela força e sabedoria concedidas ao longo desta jornada. Sua presença constante foi fundamental para a realização deste trabalho.

Aos meus pais, Albano e Deceles, e meu irmão, George, minha eterna gratidão pelo apoio incondicional, amor e incentivo que transcendem a esfera acadêmica. Vocês são meu porto seguro e inspiração.

Aos meus colegas de mestrado, Andressa, Augusto, Raul, Stefane Adna, obrigado pela amizade, companheirismo e troca de conhecimentos. Vocês tornaram essa jornada mais leve e enriquecedora.

Aos meus amigos, tanto do trabalho quanto pessoais, Andesson, Rogério, Felipe, Wesley, Rosi e demais, obrigado por acreditarem em mim, me apoiarem nos momentos desafiadores e celebrarem as conquistas comigo.

Às células de evangelização paróquial da paróquia do Cristo Ressuscitado, meu sincero agradecimento pelo apoio espiritual, compreensão e amizade. Vocês foram um refúgio de fé e encorajamento durante toda esta jornada.

Ao meu orientador, Professor Dr. Fischer Jônatas, meu profundo agradecimento pela orientação paciente, dedicada e inspiradora. Sua expertise, confiança e incentivo foram cruciais para o desenvolvimento e conclusão desta dissertação.

“Nada há de grande na terra se não está em harmonia com a eternidade.”

(Santa Catarina de Sena)

RESUMO

A investigação sobre o uso de ferramentas da indústria de testes como oráculos de geração de testes, especificamente o EvoSuite, para o apoio pedagógico, é uma área que necessita de estudos mais aprofundados. Neste contexto, foi concebido um experimento com o objetivo de avaliar o impacto do EvoSuite no processo de ensino-aprendizagem de testes de software em cursos de computação. O experimento envolveu a criação de casos de teste, com participantes divididos em dois grupos: o grupo controle, que recebeu apenas o código-fonte e a documentação, e o grupo experimental, que, além disso, recebeu uma suíte de testes gerada automaticamente pelo EvoSuite. A percepção dos alunos foi avaliada por meio de questionários aplicados em três momentos distintos: no início, durante e ao final do experimento. Os resultados indicaram que o uso do EvoSuite proporcionou a descoberta de novos cenários de teste e abordagens inovadoras, sugerindo que ferramentas automáticas de geração de testes podem contribuir significativamente para a melhoria das habilidades de teste dos estudantes e para a qualidade dos testes gerados. Em um segundo momento, foi realizado um novo estudo com o intuito de desenvolver uma ferramenta que integrasse modelos de linguagem natural, como o ChatGPT, com uma IDE online, visando oferecer aos estudantes uma plataforma prática para o aprimoramento de suas habilidades em testes de software. A proposta incluiu, ainda, o uso do PiTest para análise mutante, com o objetivo de fornecer uma camada adicional de avaliação da qualidade dos testes desenvolvidos pelos estudantes. Este trabalho é composto por esses dois estudos: o primeiro focado no uso pedagógico do EvoSuite, onde os resultados mostraram que a ferramenta auxilia na identificação de cenários de teste não previstos inicialmente, e o segundo voltado para a integração de inteligência artificial em ferramentas educacionais, proporcionando um ambiente prático e interativo para os estudantes. Ambos os estudos destacam a importância da adoção de práticas automatizadas, como testes unitários e integração contínua, além de evidenciar o papel central de habilidades interpessoais, como comunicação e resolução de problemas, na formação de profissionais qualificados. As conclusões desses estudos oferecem implicações relevantes para educadores e alunos, sugerindo que a combinação de aprendizado prático, automação e habilidades interpessoais pode aprimorar significativamente a educação em testes de software.

Palavras-chave: Testes de Software; Experimento; EvoSuite; ChatGPT.

ABSTRACT

The investigation into the use of industry tools for test generation oracles, specifically EvoSuite, for pedagogical support is an area that requires further study. In this context, an experiment was designed with the objective of evaluating the impact of EvoSuite on the teaching and learning process of software testing in computing courses. The experiment involved the creation of test cases, with participants divided into two groups: the control group, which received only the source code and documentation, and the experimental group, which additionally received a suite of tests automatically generated by EvoSuite. The students' perceptions were assessed through questionnaires administered at three distinct stages: at the beginning, during, and at the end of the experiment. The results indicated that the use of EvoSuite enabled the discovery of new test scenarios and innovative approaches, suggesting that automatic test generation tools can significantly contribute to improving students' testing skills and the quality of the tests generated. Subsequently, a second study was conducted with the aim of developing a tool that integrates large language models, such as ChatGPT, with an online IDE, providing students with a practical platform to enhance their software testing skills. The proposal also included the use of PiTest for mutation analysis, aiming to provide an additional layer of evaluation of the quality of the tests developed by the students. This work comprises two studies: the first focused on the pedagogical use of EvoSuite, where the results showed that the tool aids in identifying test scenarios not initially foreseen, and the second focused on the integration of artificial intelligence into educational tools, providing a practical and interactive environment for students. Both studies highlight the importance of adopting automated practices, such as unit testing and continuous integration, while also emphasizing the central role of interpersonal skills, such as communication and problem-solving, in the training of qualified professionals. The conclusions of these studies offer relevant implications for educators and students, suggesting that the combination of practical learning, automation, and interpersonal skills can significantly enhance software testing education.

Keywords: Software Testing; Experiment; EvoSuite; ChatGPT.

LISTA DE FIGURAS

CONTENTS

1 INTRODUCTION

In both academic and professional environments, the teaching of software testing plays a crucial role (Garousi *et al.* 2017; Setiani *et al.* 2019; Lojo e Fox 2022). In academia, this practice is essential for the solid development of students, providing essential support in all stages of the development cycle, such as planning, results analysis, and requirements evaluation. By incorporating software testing education, educational institutions empower students to tackle the challenges of the professional world.

In the professional realm, the demand for specialized professionals in this field has been continuously growing. The shortage of individuals with mastery of testing practices has resulted in low maturity in the software testing process (BENITTI 2018). Therefore, professionals skilled in this area are highly valued by companies, as they play a vital role in ensuring the quality of software products (BENITTI 2018; Perkusich 2021). As the technology industry evolves, the importance of comprehensive and updated software testing education becomes even more evident, standing out as a determining factor for the success of both students and companies aiming to develop reliable and high-performance products.

According to Silva *et al* (Perkusich 2021), the didactic and pedagogical aspects of software engineering education have, for some time, ceased to be solely an academic concern and have increasingly become present in the software industry, largely due to the industry's own demands.

Many professionals start their careers with little experience in software testing, which can be partly attributed to the gap in academic training due to deficiencies in computer science curricula. Unfortunately, in several cases, content related to software testing has been neglected (Garousi *et al.* 2020). It is essential for computer science courses to review their curricular approaches and pay proper attention to this important aspect in order to prepare future professionals more comprehensively and aligned with the market's needs (Perkusich 2021).

According to Garouse *et al* (Garousi *et al.* 2020), the global software development market projected in 2013 that from that year onwards, the costs for software quality assurance would be in the order of $312 billion annually. In the 2020 report by the Consortium for Information & Software Quality (CISQ-TI), it was noted that the costs to maintain low-quality software, only in the United States, amounted to $2.08 trillion. These significant numbers related to software quality costs reflect the importance of approaches that are more effective for teaching software testing (Aniche *et al.* 2019; Garousi *et al.* 2020).

Proposing content and teaching practices for software testing is a challenging task for teachers and tutors (Aniche *et al.* 2019; Garousi *et al.* 2020; Perkusich 2021). This is due to the need to identify and select study cases that are coherent, considering that the software used in classrooms is small and less representative compared to commercial software. The development of activities that combine theory and practice, especially with the rapid evolution of new tools and techniques, makes the task even more difficult (Aniche *et al.* 2019). To address this challenge, Benitti (BENITTI 2018) emphasizes the use of methodologies for developing learning objects focused on teaching testing. Introducing modern testing practices, such as automatic generation of test suites and techniques like mutation testing analysis, can help students better understand the techniques being adopted in the industry (Ferreira *et al.* 2018).

This work describes an experiment that evaluated students' perception in creating software test cases. Participants were divided into two groups: a control group that created test sets from scratch and an experimental group that received test sets generated by the EvoSuite tool to enhance. The experiment involved 35 undergraduate and graduate students who already had knowledge in object-oriented programming and were taking a software quality discipline. During the experiment, participants filled out forms to document their prior knowledge and note the number of tests they identified. After the practical part, the codes produced by the participants were collected and evaluated using the Pitest tool, which measured the quality of the test sets based on the number of mutants neutralized by the test suites they created.

Participants were distributed into two groups (experimental and control) and implemented tests for a given software system. The control group was tasked with creating test suites from scratch, while the experimental group received test suites generated by EvoSuite that were to be evolved. The experiment involved 35 volunteers, undergraduate and graduate students. Data collection for the experiment was carried out through the administration of three forms, applied at different stages of the experiment, with the first being the participant characterization form, the following being an experiment form, and the last being a comment form, where participants could evaluate aspects related to the experiment.

Prior to the experiment, a review class on testing concepts and techniques, as well as the tools to be used, was conducted. It is worth mentioning that participants had already taken object-oriented programming classes and were familiar with programming languages that implement this paradigm, such as Java, which was adopted for the proposed practice. During the experiment period, participants were taking a software quality course that included software

testing content in its syllabus.

During the experiment, participants were asked to fill out the form for prior knowledge, and then, starting the practical part of the experiment, to identify and write test cases for the target system of the study. The control group received the source code and its documentation. The experimental group received, in addition to the source code and its documentation, a set of test cases created by the EvoSuite tool. During the application, participants from both groups were asked to fill out the forms with information regarding their implementation, noting the quantity of tests identified.

At the end of the practical part of the experiment, the code produced by the participant was collected, along with their forms, for verification. Throughout the analysis conducted by the authors, the Pitest tool, a tool for mutant testing analysis, was used to assess the quality of the test suites produced by the participants, using the metric of the number of mutants killed by the test suites they created.

As a result, it was observed that the EVOSUITE tool can didactically support the teaching of software testing practices. Participants highlighted the utility of EVOSUITE in indicating ways to test code snippets with which the student is less familiar and even in revealing unforeseen testing scenarios (Ferreira *et al.* 2020). It was also observed, regarding the measured topics, that difficulty in implementing test cases is one of the main challenges reported by students in the experiment, as students reported more difficulties in implementing test cases than identifying test scenarios.

To connect the first study to the second, it is important to highlight that the second study was directly motivated by the findings and insights obtained in the experiment on the first one. The first study explored the application of EVOSUITE as a pedagogical tool in teaching software testing, assessing its effectiveness in an academic context. The results showed that EVOSUITE could not only facilitate the understanding of testing techniques but also introduce students to testing scenarios they might not have initially considered. This experience provided a solid knowledge base on how automated tools can be didactically integrated into software testing education.

As Figure 1 illustrates, building on this knowledge, the second study was designed to take a step further by developing a testing teaching platform called PLETES. This platform benefits from the lessons learned with EVOSUITE, integrating modern elements such as large language models (LLMs) to further enhance the software testing teaching and learning process.

By focusing on the software testing education scenario in Brazil, where there is a significant gap in the training of qualified professionals, the second study aims not only to apply the knowledge generated in the first study but also to adapt and expand teaching practices to meet the specific needs of the Brazilian market. This two-part approach, represented by 1, reflects the logical progression of the work, where the first study establishes the theoretical and practical foundation that drives the development and implementation of innovative solutions in the second study.

This work integrates two studies on software testing education. The first study evaluates the effectiveness and usability of existing software testing tools, including automatic test case generation tools like EvoSuite, in both academic and industry settings. The study emphasizes a practical experiment to provide insights into the real-world application of these tools. In contrast, the second study focuses on developing a novel testing teaching platform called PLETeS, designed to enhance the educational experience for software testing. Additionally, this study gathers opinions from students and professionals on the integration of modern artificial intelligence elements, such as Large Language Models (LLMs), into educational tools. This feedback aims to assess the current state and potential of such integration for improving the learning and teaching process.

Summary: The tool described in this paper is available at the following link: <https://github.com/gabrieldocs/pletes-monorepo>. Detailed instructions for building and running it locally, as well as enabling integration with OpenAI, can be found in the repository.

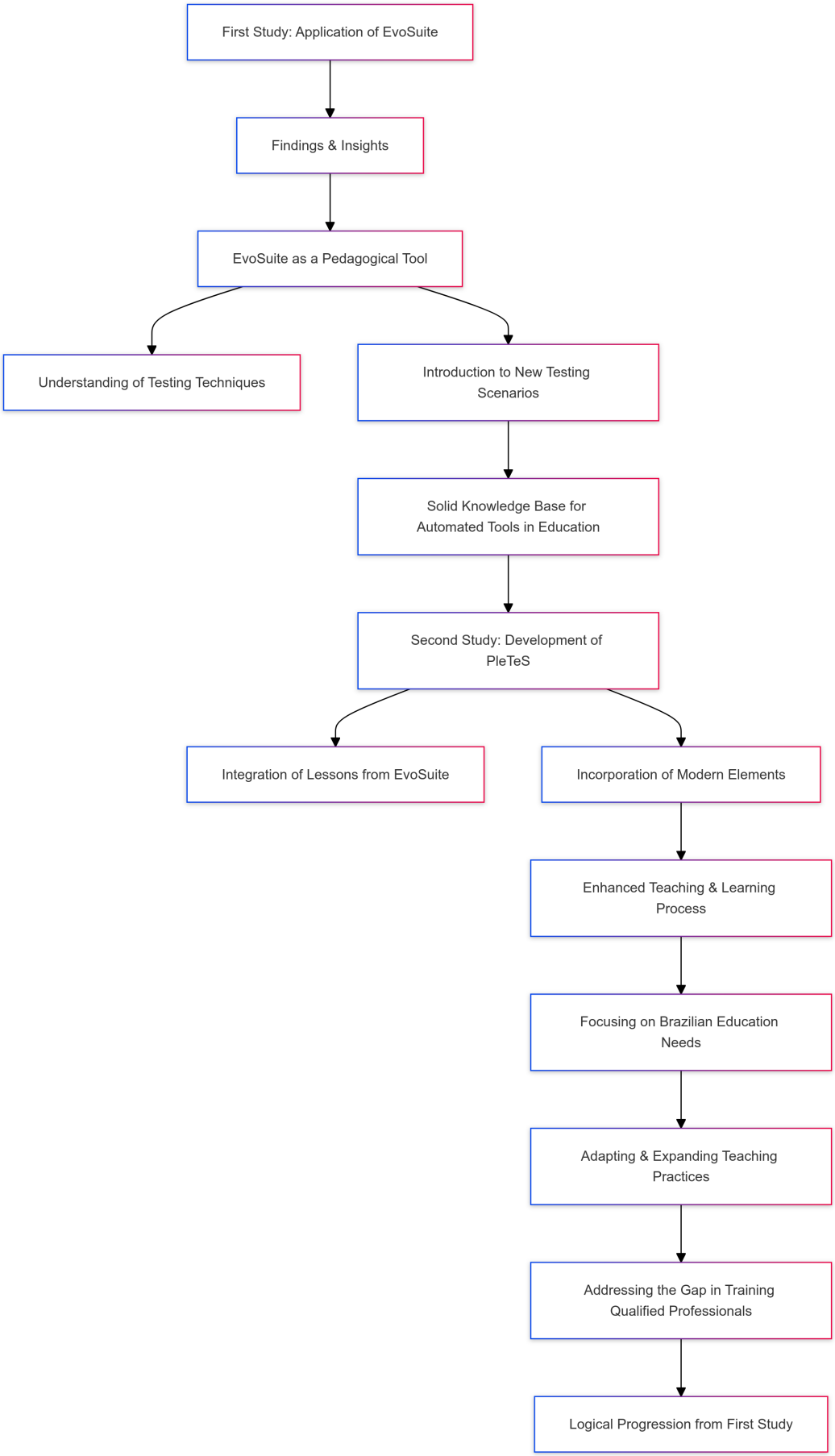

Figure 1 – Study development

2 BACKGROUND

As previously mentioned in chapter 1, this study aims to explore on the adoption of test oracles for educational purposes. This chapter describes the elements that compose the theoretical framework employed in this study. The current scenario of test education in Brazil, emerging trends on the topic and a comparison of tools for test automation is discussed on the following sections.

2.1 The scenario of software testing education in Brazil

To motivate the present work, a survey was conducted on the syllabi of computing courses at each of the Brazilian federal universities, searching for courses in computing that include disciplines focused on software testing. Out of the 144 federal universities in Brazil surveyed, only 44 (30.55%) universities offer any discipline focused on testing. Thus, 100 (69.44%) Brazilian federal universities with computing courses have no specific software testing discipline. Table 1 displays the data found for each federal university across all brazilian states, on how many campi have at least one computer-related course and how many software testing subjects there are on the university. The complete list of federal universities with their respective software testing courses is available on the website of this work [1]. In fact, many universities neglect testing activities in their curricula, which can directly reflect on the difficulty graduates face in dealing with software testing in the software industry (Edwards 2004; Gopal *et al.* 2021). As exposed, in many courses, training on software testing is absent or insufficient (Valle *et al.* 2015; Paschoal e Souza 2018).

The creation of specific content for software testing education is of utmost importance for the development of qualified professionals and for the quality of software products delivered to the market (Garousi *et al.* 2020). Testing is a critical step in the development cycle, as it ensures defect detection, compliance with requirements, and software robustness (Ammann e Offutt 2016). By developing content focused on this topic, it is possible to address in more depth the various testing techniques, validation strategies, and available tools, allowing students to acquire specific and up-to-date knowledge. Moreover, by directing education towards this area, educational institutions and companies in the sector contribute to the formation of a quality culture, promoting error reduction, increasing cus-

[1] https://organizationstart.github.io/first/

State	Quantity of disciplines offered	Number of campi
Acre	0	1
Alagoas	2	3
Amapá	0	1
Amazonas	2	5
Bahia	0	4
Ceará	2	6
Distrito Federal	1	5
Espírito Santo	0	3
Goiás	3	6
Maranhão	0	2
Mato Grosso	0	3
Mato Grosso do Sul	4	7
Minas Gerais	7	18
Pará	1	5
Paraíba	1	4
Paraná	5	6
Pernambuco	1	6
Piauí	1	2
Rio de Janeiro	3	8
Rio Grande do Norte	4	10
Rio Grande do Sul	2	17
Rondônia	0	1
Roraima	0	1
Santa Catarina	1	3
São Paulo	1	12
Sergipe	3	3
Tocantins	0	2
Total	44	144

Table 1 – Quantity of computer-related courses per state of Brazil per federal university and test subject per university

tomer satisfaction, and consequently building a more reliable and competitive software industry (Valle *et al.* 2015; Paschoal e Souza 2018; Gopal *et al.* 2021).

The adoption of software development industry practices by academia has become increasingly important and beneficial. By bringing these practices into the academic environment, students have the opportunity to become familiar with the practices and tools used in the industry, preparing them more effectively for the job market. This means that undergraduates acquire practical skills and applied knowledge, as well as a more realistic understanding of the challenges and demands of their future professional field.

Furthermore, the adoption of software development industry practices by academia contributes to collaboration and knowledge exchange between the two spheres. The industry

can benefit from engagement with academia, gaining insights and innovative research that can drive technological progress. In turn, academia can benefit from the practical experience and perspective of industry professionals, enhancing their curricula and developing research that addresses real market needs. This partnership between academia and software development industry creates a cycle of continuous and enriching learning, driving innovation and advancement in the field of software development.

2.2 Emerging trends in software testing education

In recent years, the field of software testing has undergone significant evolution, driven by advancements in technology and changing industry demands. Consequently, the landscape of software testing education has also experienced notable shifts, with emerging trends reshaping the way testing concepts are taught and learned (Garousi *et al.* 2017; Lojo e Fox 2022).

One prominent trend in software testing education is the increasing emphasis on practical, hands-on learning experiencess (Perkusich 2021). Traditional lecture-based approaches are being supplemented, if not replaced, by interactive workshops, simulations, and real-world case studies. This shift recognizes the importance of experiential learning in developing the skills necessary for effective software testing. By engaging students in practical exercises, educators can better prepare them for the challenges they will encounter in actual testing scenarios, fostering a deeper understanding of testing principles and techniques.

Another key trend is the integration of automation and artificial intelligence (AI) into software testing curricula (Huizinga e Kolawa 2007; Leppänen *et al.* 2015; Ammann e Offutt 2016; Shahin *et al.* 2017). As automation becomes increasingly pervasive in the software development lifecycle, proficiency in tools and techniques for automated testing is becoming essential for software testers. Accordingly, educational programs are incorporating modules on test automation frameworks, scripting languages, and AI-driven testing methodologies. By familiarizing students with these technologies early in their education, institutions are better equipping them to adapt to the evolving demands of the industry and remain competitive in the job market.

Furthermore, there is a growing recognition of the importance of soft skills in software testing education. Beyond technical proficiency, effective communication, teamwork, critical thinking, and problem-solving abilities are crucial for success in the field (Hazzan e Har-Shai 2013). As such, modern software testing programs are placing greater emphasis on the development of these soft skills through collaborative projects, group discussions, and presentations. By

nurturing well-rounded professionals who can excel not only in technical tasks but also in interpersonal interactions, educational institutions are preparing students to thrive in diverse and dynamic work environments.

In conclusion, software testing education is undergoing a transformation driven by emerging trends that reflect the evolving nature of the field. By embracing hands-on learning, integrating automation and AI, and fostering the development of soft skills, educational programs are better equipping students to navigate the complexities of modern software testing(Edwards 2004; Gopal *et al.* 2021).. As these trends continue to evolve, educators must remain agile and adaptable, ensuring that their curricula remain relevant and effective in preparing the next generation of software testers for success.

2.3 Unit testing and test automation

Unit tests are designed to evaluate the smallest units of code, checking if the requirements are being met. They are developed during the implementation stage and constitute the most basic level of testing (Ammann e Offutt 2016). The units they refer to can be methods, functions, classes, or modules.

On the other hand, automated tests are software tests that can be executed through automation mechanisms or artifacts. Automation of test execution is essential for continuous integration software development practices (Huizinga e Kolawa 2007; Leppänen *et al.* 2015; Shahin *et al.* 2017), so the adoption of tools capable of accelerating this process has been a subject of interest in academia and industry.

Listing 1 – Implementation of method compraNoCredito

```
1
2  package example.app.models;
3
4  public class Carteira {
5
6      public void compraNoCredito(double valor) {
7          try {
8              if(this.limiteCredito >= valor) {
9                  this.limiteCredito = valor;
10             } else {
11                 throw new Error(
12                 "Saldo insuficiente"
13                 );
14             }
15         } catch(Error e) {
16         System.out.println(e.getMessage()); }
17     }
18 }
```

Listing 1 presents the implementation of the "compraNoCredito" method belonging to the "Carteira" class in a Travel system. This method is responsible for allowing a customer to make a purchase on credit, where the "value" parameter represents the value of the purchase to be made. The method uses a conditional control structure "if" to check if the "creditLimit" available in the wallet is greater than or equal to the purchase value. If this condition is met, the method updates the "creditLimit" by subtracting the purchase value, effectively deducting the value from the available credit. However, if the "creditLimit" is less than the purchase value, the method throws an exception of type "Error" with the message "Saldo insuficiente". In case of an exception, the error message is printed to the console.

It is important to note that the exception code is inside a "catch" block that catches exceptions of type "Error". However, exception handling in this code is limited to printing the error message to the console without taking additional measures, such as reversing the operation. Therefore, this method handles credit purchases but does not provide a comprehensive approach to exception handling or proper transaction logging.

2.4 Comparison of automated testing Tools

Automated testing tools play a crucial role in the software testing process, enabling testers to efficiently generate test cases, execute tests, and analyze results. Several automated testing tools are available, each offering unique features and capabilities. In this subsection, for the Java language and ecossystem, we compare four popular automated testing tools: EvoSuite, Randoop and Pitest.

Each tool has been evaluated based on several criteria, including the need for installation, the requirement for scripting knowledge, access to code, user-friendliness, availability of tutorials, the ability to generate test suites, provision of test result reports, playback functionality, identification of tested objects, and support for plugins and extensions. This comparison serves as a valuable resource for testers seeking to select the most suitable automated testing tool for their specific requirements.

Based on the comparison table provided (Table 2), EvoSuite emerges as the preferred tool for working with the experimento, Java 8, and the Netbeans ecosystem for several reasons:

Criteria	EvoSuite	Randoop	Pitest
Install Required	Yes	Yes	Yes
Knowledge of scripts required	No	No	No
Access to code required	Yes	Yes	Yes
User friendly interface	Yes	Yes	No
Tutorial on how to use	Yes	Yes	Yes
Generate test suite	Yes	Yes	Yes
Test result reports	Yes	Yes	Yes
Playback	No	No	No
Identifying the tested object	Yes	Yes	Yes
Plugins and extensions	Yes	No	No

Table 2 – Comparison of Java Testing Tools

1. **Install Required**: EvoSuite requires installation, which aligns with the requirement for any tool to be used effectively within the Java 8 and Netbeans ecosystem.
2. **Knowledge of scripts required**: EvoSuite stands out as it does not require knowledge of scripts, unlike Klee which does. This makes EvoSuite more accessible to a wider range of users, including those without extensive programming experience.
3. **Access to code required**: EvoSuite, like the other tools, necessitates access to code. This aligns with the nature of automated testing tools, which typically analyze code to generate test suites.

4. **User-friendly interface**: EvoSuite is noted for its user-friendly interface, which enhances usability and makes it easier for testers to navigate and utilize its features effectively. This is particularly advantageous for users within the Netbeans ecosystem, where ease of use is valued.
5. **Tutorial on how to use**: EvoSuite offers tutorials on how to use the tool, which is essential for onboarding users and ensuring they can effectively leverage its capabilities. This facilitates smoother integration into the experimento and the broader Java 8 and Netbeans environment.
6. **Generate test suite**: EvoSuite, like the other tools, is capable of generating test suites. This is a fundamental feature required for automated testing within the Java 8 and Netbeans ecosystem.
7. **Test result reports**: EvoSuite provides test result reports, enabling testers to analyze the outcomes of test executions effectively. This feature is crucial for identifying issues and assessing the quality of the software being tested.
8. **Playback**: Although EvoSuite does not offer playback functionality, this may not be a critical requirement for the experimento, Java 8, and Netbeans ecosystem, where the focus is likely more on test generation and result analysis rather than playback of test executions.
9. **Identifying the tested object**: EvoSuite, along with Randoop, supports identifying the tested object, which enhances the clarity and understanding of test results within the experimento and the broader Java 8 and Netbeans context.
10. **Plugins and extensions**: EvoSuite supports plugins and extensions, offering flexibility for customization and integration with other tools or frameworks within the Java 8 and Netbeans ecosystem. This adaptability is valuable for tailoring the testing process to specific project requirements.

In summary, EvoSuite was adopted as the preferred automated testing tool for the experiment, Java 8, and the Netbeans ecosystem due to its combination of features such as user-friendliness, accessibility, robust test suite generation, comprehensive test result reporting, and support for plugins and extensions.

2.5 Test case generation with EVOSUITE

Automatic test case generation tools have become increasingly relevant in the context of software development as they offer an efficient way to increase test coverage

and identify potential issues in the code. Among these tools, notable ones include EvoSuite (Fraser e Arcuri 2011), Randoop (Pacheco e Ernst 2007), and Toradocu (Goffi *et al.* 2016). EvoSuite, chosen for this scenario, is a widely recognized tool for its ability to automatically generate high-quality test cases in Java. Utilizing heuristic search techniques (Vogl *et al.* 2021), EvoSuite explores various combinations of execution paths to create tests that cover multiple scenarios, aiming to optimize code coverage.

With its comprehensive approach and efficiency in test generation, EvoSuite is a promising choice to support software testing education and provide students with hands-on experience in creating efficient and reliable test cases. Among its key features, one can mention firstly its ability to generate tests for the Java language compatible with the JUnit 4 framework[2], in addition to being open-source, facilitating further studies and research. The code in Listing 2 is an example generated by the EvoSuite tool.

Listing 2 – Example generated by EvoSuite

```
1  @Test(timeout = 4000)
2  public void test13()  throws Throwable  {
3    Carteira carteira0 = new Carteira(
4      (-592.54), (-592.54), (-592.54)
5    );
6    carteira0.compraNoCredito((-1.0));
7    assertEquals(
8      (-1777.62),
9      carteira0.saldoTotal(),
10     0.01);
11 }
```

In this specific example, the code is a unit test case, indicated by the "@Test" annotation at the beginning of the method. The test method is named "test13" and was generated by EvoSuite to test the "Wallet" class. This class represents a virtual wallet with functionalities related to purchases and balance.

In the body of the test method, we have the following actions: on line 4, an instance of the "Carteira" class is created with three initial balance values, all set to "-592.54". On line 5, the "purchaseOnCredit" method of the "Wallet" class is invoked with the value "-1.0"

[2] https://junit.org/junit4/

as a parameter. This method simulates a purchase made with credit payment. On line 6, the "assertEquals" method is used to verify if the total balance of the wallet after the purchase is equal to "-1777.62". The third parameter "0.01" is a tolerance for comparing floating-point values, allowing a small difference in results due to calculation inaccuracies.

The objective of this test case is to verify if the "purchaseOnCredit" method is functioning correctly in updating the total balance of the wallet after a purchase. The EvoSuite tool generates this test case automatically, exploring different input and execution scenarios to ensure a more comprehensive coverage of the tested code.

3 RELATED WORK

This chapter presents a comprehensive review of related work in the field of software testing education, encompassing both the use of automated tools and the integration of innovative pedagogical approaches.

3.1 Landscape of software testing education

In comparison to some related works, this study stands out for the use of the EVOSUITE tool as pedagogical support for software testing education. For example, the work by Corte *et al*. (Corte *et al.* 2007) discusses the use of the PROGTEST software to automate the evaluation of programming and software testing assignments. Although the PROGTEST tool is capable of assessing the quality of code and tests submitted by students, it may not be sufficient to help them identify suitable test scenarios for the code in question. In contrast, the current work utilizes EvoSuite to support students in identifying these test scenarios. EvoSuite is capable of automatically generating high-quality test cases, allowing developers to have an initial suite of automated tests at a low cost. This experience report investigates whether EvoSuite can also serve as pedagogical support, indicating new test scenarios and ways to test code snippets with which students may struggle.

Some works, for example, Benitti and Draylson (BENITTI 2018), focus on creating teaching materials and methodologies that clearly and objectively address the fundamentals of testing, using automation mechanisms to present effective solutions for teaching these topics. On the other hand, the current work aims to verify if students can use the EvoSuite tool to broaden their understanding and perspective on the various possible tests. The work seeks to provide stakeholders with an innovative approach, allowing them to explore new test scenarios and discover additional ways to test code snippets, thus enriching the software testing teaching and learning process.

The study "ProTesters: a board game for teaching the testing process" by Gonçalves et al. (2022) (Moreira *et al.* 2023) addresses the issue of student motivation and uses gamification strategies to teach software testing. This contrasts with the current work, which follows a more traditional software testing methodology, preparing students for a development environment similar to what they will encounter in the software industry. On the other hand, the study "How do students test software units?" by Bijlsma *et al*. (2021) (Bijlsma *et al.* 2021) explores the

beliefs and testing practices of newly graduated programming students, aiming to understand the approaches they will take when creating software tests. The present work seeks to understand the perceptions and approaches of students who, now familiar with software testing tools and techniques, face the task of identifying and creating software tests, as well as evaluating their benefit in using test suites generated by EvoSuite, with the aim of expanding existing test suites.

In general, articles related to software testing education and automation tools seek to present effective solutions for teaching these topics, contributing to the training of more skilled professionals prepared to work in the software development field (Souza *et al.* 2012; Garousi *et al.* 2020; Perkusich 2021). The current work aligns with this proposal, using EvoSuite as a tool to support software testing education, providing a differentiated approach and exploring automated test case generation to enhance students' understanding of the topic.

In a Ouh et al study (Ouh *et al.* 2023), researchers explored using the ChatGPT language model to generate solutions for coding exercises in a Java programming course. They found that ChatGPT accurately produced Java code with good readability and organization for various exercises, offering alternative, memory-efficient solutions. However, the model struggled with exercises containing non-textual descriptions, leading to invalid solutions. Despite limitations, ChatGPT shows promise as a helpful tool for students navigating programming challenges. Educators can use these findings to design exercises that maintain integrity while leveraging ChatGPT's assistance. This study informs our own research by providing insights into ChatGPT's capabilities for generating test cases, offering valuable references for students using our platform.

Benitti and Draylson (BENITTI 2018) emphasized teaching materials and methodological practices for automation artifacts in software testing fundamentals. In comparison, our work focuses on developing a platform that facilitates the teaching-learning process of software testing. We aim for an innovative approach, allowing exploration of new testing scenarios with Large Language Models (LLMs) (Schäfer *et al.* 2024) support, expanding student opportunities.

Gonçalves *et al.* (Moreira *et al.* 2023) addressed student motivation through gamification. In contrast, our work follows a software testing methodology with playful elements, motivating students to explore potential test cases with LLMs support. The study by Bijlsma *et al.* (Bijlsma *et al.* 2021) focused on recent computing course graduates, investigating their knowledge and experiences in creating test cases.

In summary, papers related to teaching software testing and automation tools aim to

present effective approaches for instruction, contributing to the development of better-prepared professionals in software development and testing (Garousi *et al.* 2020). Our work aligns with this objective, utilizing the PLETES platform as a software testing teaching tool. It provides a differentiated approach, exploring the generation of test hints and even code fragments via artificial intelligence.

4 FIRST STUDY EXPERIMENT DESIGN

This chapter presents the overview of the experiment on EVOSUITE usage for pedagogical purpose. Section 4.1 discusses the research questions (RQs) of this study. Section 4.2 provides an overview of the proposed experiment. The forms related to the data generated during the participants' prior knowledge and professional experience, applied during the experiment's stages, are addressed by Section 4.3. The experiment's stages are covered in Section 4.4. The participant set is addressed in Section 4.5, and the group division in Section 4.6. Finally, Section 4.7 addresses the experiment's execution environment.

4.1 Objective and research questions

The present study has the following objective (Basili e Rombach 1988): *to analyze* the perception of students when dealing with software testing activities; *with the purpose of* identifying students' difficulties; *with respect to*: i) the use of EVOSUITE for automated test generation; ii) difficulties reported by participants in identifying and writing new tests; iii) correctness of written test cases; *from the perspective of* students *in the context of* undergraduate and graduate students in computer science and software development courses. To achieve these objectives, four research questions (RQs) have been defined as follows:

RQ1*: Can the EvoSuite automated test generation tool be used for didactic purposes?* This research question seeks to verify the feasibility and effectiveness of using EvoSuite as an educational tool to teach automated test generation and demonstrate its practical applications in software development.

RQ2*: Which experiment group identified and wrote more effective tests?* This research question aims to evaluate the presumed benefits that the use of EvoSuite can bring to the participants.

RQ3*: What factors can hinder the implementation of the test suite?* This research question aims to understand participants' perception regarding the tasks proposed by the experiment and what other elements are related to the difficulties faced in completing the tasks.

RQ4*: Does prior experience with testing practices influence the studentt's effectiveness of test case creation and identification?*This research question aims to explore the influence of participants' prior experience with automated testing tools on their ability to identify and create effective test cases on their perspectives.

4.2 Overview of the experiment

In the study, 35 undergraduate and graduate students participated. They were divided into two groups: one group that had to create tests from scratch (control group) and another group that received ready-made tests and needed to improve them (experiment group). Before the study, they all received a review lesson on testing and the tools to be used. During the study, participants filled out three forms at different times: the first for personal characterization, the second about the experiment itself, and the third to provide feedback.

The participants, who already had knowledge in object-oriented programming and were familiar with languages like Java, had to identify and write test cases for a system. The control group received only the source code and documentation, while the experiment group received these materials plus some test cases generated by the EvoSuite tool. They filled out forms with information about their tests, including the number of tests identified. At the end of the experiment, the code they produced, along with the filled-out forms, was collected and analyzed using the Pitest tool, which measured the quality of the tests based on the number of errors they found in the code.

4.3 Experiment artifacts

The following artifacts were necessary for the execution of the experiment. To assist in future replications of the present study, all artifacts used are available on the project's website[1]. ***Experiment Forms***: *Consent*: This document describes the objectives of the experiment, requests authorization for the use of the data generated for analysis, and ensures anonymity upon publication of the data. *Prior Knowledge*: This document contains five questions about the participant's prior knowledge, covering their knowledge of the programming language adopted, their professional experience, and their knowledge of the experiment's topic. *Experiment*: This document provides the student with instructions on the order of execution of the experiment activities and the appropriate space for annotating the tasks.
Experiment Target System *Travel Agency*: A system developed in Java language, for which the participant will write unit tests. The system consists of nine classes and two enumerations.

Listing 1, presented earlier, demonstrates a fragment of the **Company** class, which is part of the system provided to the participant groups [2].

[1] https://organizationstart.github.io/first/
[2] https://github.com/organizationstart/first

Development environment settings: *Eclipse* was used as the IDE for Java programming, along with *Java JDK 8*. *Apache Maven*: tool for project dependency management. *JUnit 4.12*: Java testing framework used for the experiment. *EvoSuite 1.0.6*: tool for automatic test suite generation. *Pitest*: tool for mutation testing.

4.4 Experiment stages

Figure 2 presents the three stages conducted in the proposed experiment: participant allocation, test suite implementation, and test suite effectiveness analysis.

Participant allocation or Laboratory: Participants were allocated to computers prepared for the execution of activities, which also contained the forms to be filled out. Various data were collected during this stage regarding professional experience time, affinity level with the Java programming language, and the JUnit testing framework. Participants were also asked about their level of experience in reading UML diagrams, reading requirements, and other practices related to documentation interpretation. During this stage, the objectives and scope of the tasks to be performed and the provided instruments were presented.

Test suite implementation or Experiment: In the second stage of the experiment, the class was divided into two groups: the control group (CG), which received a version of the project without unit test files, and the experimental group (EG), which received the project with unit tests already generated using the EVOSUITE tool, with predefined initial coverage. Participants were randomly and evenly distributed among control and experimental groups. The provided development environment was ECLIPSE[3]. Additionally, in the second stage, participants from both groups were asked to list the identified test cases for the system and attempt to implement the test cases.

Test suite effectiveness analysis or Evaluation: At the end of Stage 2, the data generated by participants from both groups were collected and could proceed to the analysis stage. Initially, it was necessary to standardize the data to remove outliers and broken tests. In particular, the goal was to understand whether having experience in the job market as a developer impacts the quantity of tests created by the participant and the effectiveness of this test suite.

[3] https://www.eclipse.org/downloads/

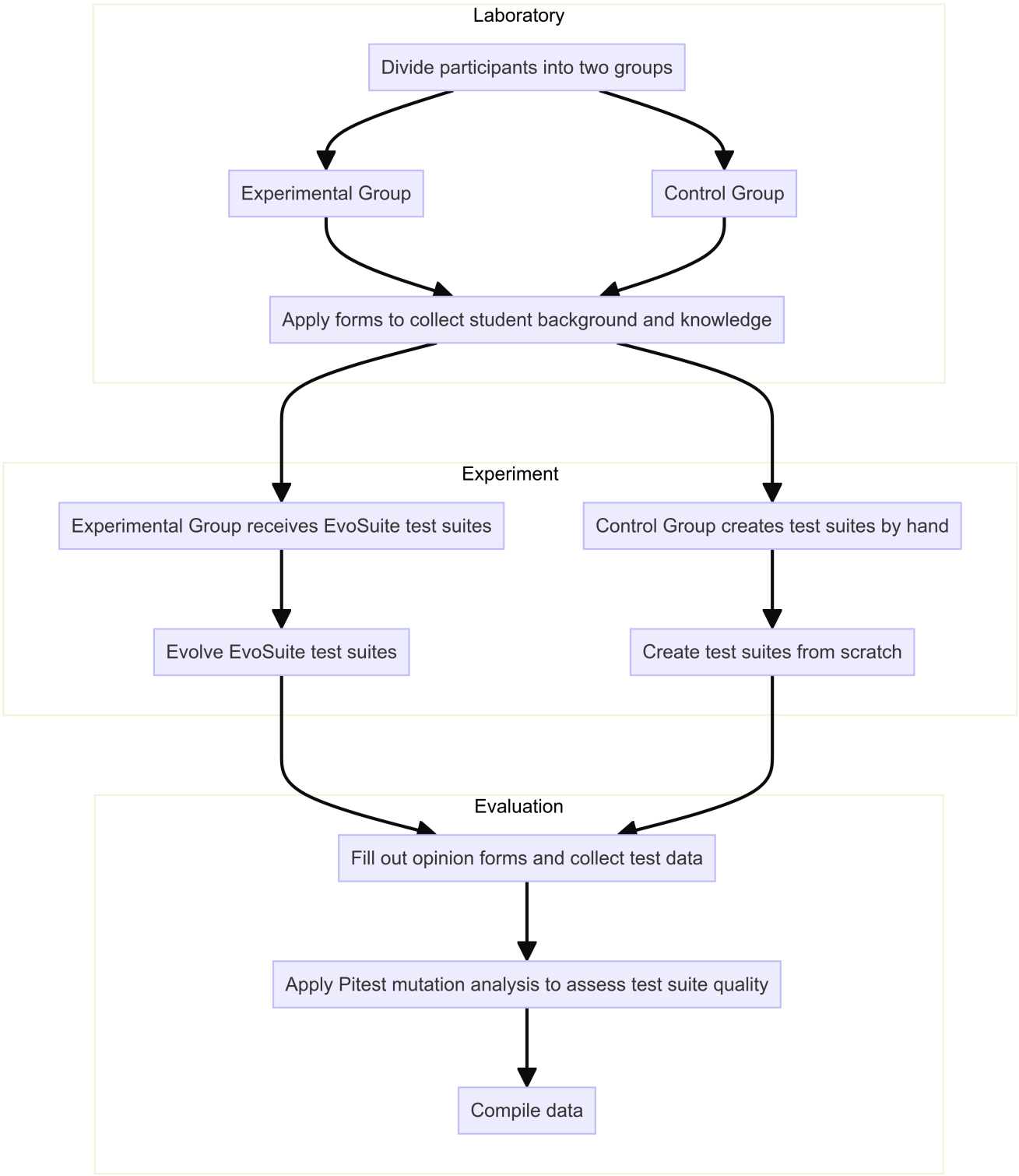

Figure 2 – Experiment methodology

4.5 Participant group

The initial participant group consisted of a total of 44 students, with 41 undergraduate students in computer engineering and 3 students from the Graduate Program. Both courses are taught at a university where the experiment was conducted as part of the software quality

discipline (undergraduate and graduate) during the second semester of the year 2022. Out of the total initial participants, 9 were excluded from the study, leaving 35 participants with valid data, including 3 graduate students and 32 undergraduate students. Participants were excluded from the study due to 7 participants not completing the experiment forms and 2 participants providing forms with unprocessable data.

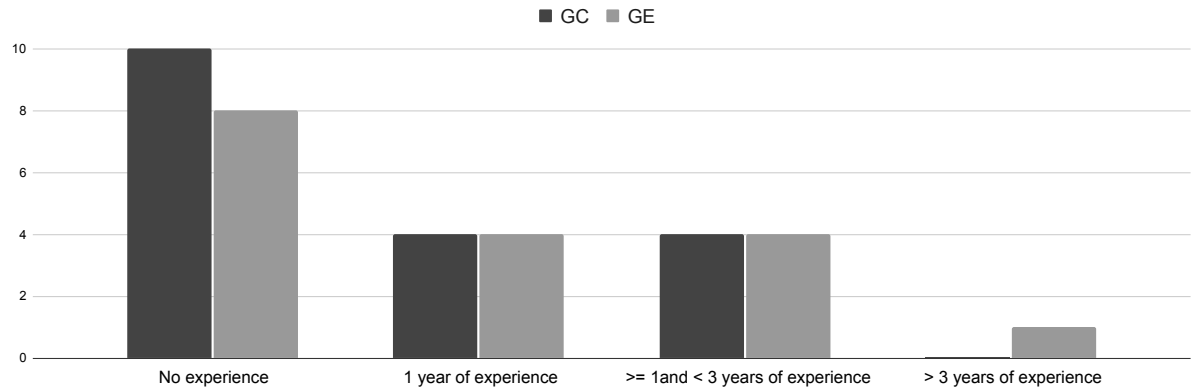

Figure 3 – Answers to the professional experience level form

Figure 3 displays the level of professional experience reported by the participants, with 0 marked if the participant had never worked professionally as a developer, 1 if they had up to 1 year of professional experience, 2 if they had 1 to 3 years of experience, and 3 if they had worked more than three years in the job market. As observed, out of the total participants, 18 declared no previous professional experience while 17 declared having had some level of professional involvement.

4.6 Allocation of participants to groups

Participants in this study were randomly divided into two distinct groups: the control group (CG) and the experimental group (EG). This random division ensured that both groups were representative and comparable in terms of participants' characteristics and skills. The random division approach also minimized selection bias and increased the reliability of the obtained results.

Furthermore, the laboratory where the practice was conducted was physically divided into two sides to accommodate the control and experimental groups separately. Each side of the laboratory was equipped with computers prepared with the files and resources necessary for the proposed practice. The side designated for the control group contained the source code of

the software system to be tested and the relevant documentation. On the other hand, the side designated for the experimental group, in addition to having the same material as the control group, also provided a test suite generated by the EvoSuite tool. This physical configuration allowed the groups to have access to resources specific to their respective activities, ensuring the integrity and reliability of the comparison between the two groups during the experiment.

4.7 Execution environment

The project was configured through Maven, a project management and comprehension tool for software projects. Based on the concept of a project object model (POM), Maven can manage the build, reporting, and documentation of a project from a central piece of information. The installation of Maven on the operating systems used (Linux and Windows) is similar and is detailed on its access page. The pom.xml file is located in the root directory of the project, containing information about the project configuration, such as dependencies and plugins that would be used for Maven to execute, and it is in this file that information about JUnit, EvoSuite, and JavaDoc, which are necessary technologies for the experiment's execution, is located.

For the execution of the experiment, it was necessary to install and configure the testing mechanisms belonging to the JUnit suite and the dependencies required for its execution, as well as the tool for generating test cases automatically. After installing the dependencies, the following steps were taken: i) importing the project into the Eclipse IDE; ii) verifying the original code coverage to confirm the absence of test artifacts that could interfere with EvoSuite; iii) executing the tool to generate test cases via the console, through Maven.

A copy of the test cases generated by the tool was provided to each participant, with a pre-established coverage level. The project was configured so that the initially provided coverage, although high, allowed participants to focus solely on identifying additional test cases and implementing them when necessary.

5 RESULTS OF THE EXPERIMENT

This chapter presents the results obtained and the analysis conducted after the experiment was applied. Section 5.1 discusses the use of EVOSUITE as educational support. Section 5.2 then debates which group was more effective in writing tests, and Section 5.3 seeks to present the factors that hindered implementation by the participants. Section 5.4 discusses participants prior experiences influence on the participation.

5.1 Can the automated test generation tool EVOSUITE be used for educational purposes? (RQ1)

The reason for the creating the two groups (control and experimental) to address the research question related to the EVOSUITE tool for educational purposes was to verify if participants could implement tests in the provided system. Through the data from the control group, it was indicated that indeed, it was possible. Even without any initial test cases, participants were able to implement a test suite. However, the main focus of RQ1 is to observe the behavior of the experimental group in dealing with the test suite provided by EVOSUITE. Thus, it was verified whether EVOSUITE can provide educational support for students to expand the test suite with new test cases. The challenge for both groups was quite distinct because even though they had to implement the test suite for the same system, the control group had no initial tests to evolve, while the experimental group had a comprehensive test suite that needed to be partially understood in order to build new tests.

It was observed through the experiment data that the control group implemented a greater number of unit tests. On average, 5.3 tests were implemented. However, the experimental group implemented only an average of 1.1 unit tests. The difference in the average number of tests implemented by the two groups is mainly explained by the fact that the experimental group previously had a set of tests provided by the EVOSUITE tool. Therefore, participants in the experimental group had to make a greater effort to find a test scenario that had not yet been covered by the tool. However, participants in the control group, as they had no previously implemented tests, could start by implementing the most basic cases, such as constructor methods and standard *get* and *set* methods. This result was already expected, but this evaluation was necessary to verify if the experimental group, even with a previously provided suite, was capable of evolving it. Through the provided results, it was observed that yes, participants in

the experimental group were able to evolve the provided test suite. Additionally, participants indicated that the EVOSUITE tool helped to show test scenarios that they had not thought of, as well as indicating ways to access code points they did not know how to reach. Thus, from the obtained data, there is strong evidence that the EVOSUITE tool can be used in an educational manner for the implementation of test cases.

The following question was asked to the participants of the experimental group: *"Do you believe that the* EVOSUITE *tool can facilitate the process of writing test code for a system? Justify your answer."* Some responses from the participants collected in the forms indicate that EVOSUITE can provide educational support, as follows: *"Because the EvoSuite tool brings with it the possibility of creating test cases automatically, it assists the developer with an initial boost in test development."* Furthermore, a second participant mentioned the following: *"Because the tool generates automatic test codes. Thus, someone who does not have much knowledge of software testing can use it."* Additionally, a third participant said: *"Certainly. This tool already creates test cases ready to be used, having good code coverage, reducing the developer's time to create tests."* However, there were reports from participants who identified difficulties in dealing with the test suites generated by the tool. As follows: *"The tool is a great ally to help the tester think about non-trivial cases; however, it can make it difficult to understand which tests have already been performed and which are still needed."*

Although this result seems promising, a more extensive investigation and monitoring of the use of EVOSUITE for educational purposes over a longer period of time are necessary. It was observed that the EVOSUITE tool can help participants increase their understanding of the source code of the experimental system. Since participants did not know how to access certain points of the code through testing, through the test cases generated by EVOSUITE, it was possible to have an indication of how to do so. This was, according to them, essential for understanding the code of the target system and creating new test cases. Thus, both teachers and students can benefit from this result and use the tool for educational purposes.

5.2 Which group wrote more effective tests? (RQ2)

Analyzing tests solely based on quantity is not an effective measure. It is necessary to assess the effectiveness of the tests, whether they are truly capable of detecting faults. To achieve this, the technique of mutation testing was used in this study. This technique allows for the artificial insertion of faults, corresponding to real faults (Petrović *et al.* 2022; Sánchez *et al.* 2022).

Thus, each test suite implemented by the study participants was evaluated with mutants generated for the target system of the experiment. The mutation testing technique was not used by the participants during the experiment, so they could not perform tests aimed at killing mutants. However, to assess the effectiveness of the tests implemented by the participants of both groups, the metric adopted was the number of mutants killed during mutant testing analysis. In this sense, the authors of this study meticulously examined each set of tests developed by the participants and, through the tool called PITESTE[1], evaluated the number of inactive mutants for each test suite.

For the target system of the experiment, the PITESTE tool generates 139 mutants. When individually running each test suite from the control group, it was observed that on average 4% of the mutants were killed. However, for the experimental group, only the tests created without the test suite generated automatically by EVOSUITE had an average percentage of 35.5% mutants killed. The test suite generated by EVOSUITE without additional participant tests is capable of killing 35% of the mutants. Thus, the experimental group did not show a considerable improvement for the test cases generated. However, it was observed that the tests created by participants who used EVOSUITE were more logically exercising the target system compared to participants who did not use the tool. This result indicates that the quality of the test cases generated by the group that used EVOSUITE had a considerable improvement compared to the other group that did not use it.

5.3 What factors can difficult the implementation of the test suite? (RQ3)

Alongside the analysis of data from the current experiment, a survey was conducted on the syllabi of computer science courses at each of the Brazilian federal universities in search of courses in computing that have disciplines focused on software testing. Out of the 143 federal universities in Brazil surveyed, only 44 (30.7%) universities have any disciplines focused on testing. Thus, 99 (69.3%) Brazilian federal universities, with some computing course, do not have any specific software testing discipline. The complete list of federal universities with their respective software testing courses is available on the website of this work. This result can be pointed out as one of the main difficulties in teaching testing. In fact, many universities neglect testing activities in their curricula, which directly reflects the difficulty that students face in dealing with software testing. As exposed, in many courses, training on software testing is absent

[1] https://pitest.org/

or insufficient.

With the aim of identifying factors that may hinder students' performance in testing activities, participants in the experiment were asked about several topics related to testing. They were requested to indicate the level of difficulty they faced in the following areas: Java language, system requirements identification, understanding requirement specifications, test case identification, handling JUnit, and reading UML diagrams. These topics were selected based on related works (Mendes *et al.* 2019; Jr *et al.* 2021; Zanetti *et al.* 2023). To allow participants to indicate if they faced any difficulty related to these topics, they answered the questionnaire after completing the test implementation tasks. Responses were given on a closed interval scale from 1 to 5, mapped to the Likert scale (Joshi *et al.* 2015), ranging from *little difficulty (1)* to *much difficulty (5)*. It's important to emphasize that for each topic, participants could choose only one response from 1 to 5.

Figure 4 – Distribution of difficulty levels

Figure 4 presents the distribution of difficulty levels across the topics: *Java language*, *identifying system requirements*, *understanding requirements specifications*, *identifying test cases*, *handling JUnit*, and *reading UML diagrams*. Figure 4 shows the highest concentration of responses was for the option *little difficulty (1)*, where for the topic *Reading requirements* and *Handling JUnit* obtained 14 and 13 participants, respectively. For other difficulty levels, for example, for the topic *Reading requirements*, the following results were found: one participant reported *some difficulty (2)*, four participants reported *moderate difficulty (3)*, five participants reported *some difficulty (4)*, and three participants reported *much difficulty (5)*.

Furthermore, a highlight is for the Java topic, which has high values for *no difficulty (1)* but has eight participants who reported *much difficulty (5)*. This result is consistent with some responses written by participants who report being able to design the test case but have difficulty implementing it. For example, it was asked: *Was implementing tests with JUnit an easy task?* responses in this sense were: *"In my case I had difficulties, but it is solely due to lack of*

familiarity with the programming language". Additionally, a second participant mentioned: *"No, but it wasn't because of the tool, but rather my difficulty in writing the tests and also because I also have little knowledge in JAVA programming."* But, it can also be observed that identifying the test case is one of the topics highlighted in the distribution of difficulty levels presented in Figure 4. As an example, here is a participant's response in this regard: *"It was not an easy task, as I had difficulties in identifying the test cases"*.

From this result it is possible to perceive the concepts of test case implementation during the course seemed to be well consolidated for the students, as they were able to identify the test scenarios in the experiment.

Participants in the control group identified an average of 14.4 test scenarios, implementing an average of 5.3 test cases. Participants in the experimental group identified 14.3 test scenarios, a very close average to the control group, but implemented an average of 1.1 test cases. During the experiment execution, according to the participants' reports, the main difficulty was in implementing the test cases. Even though the students were familiar with the Java language and it was used in several courses in the curriculum, the students reported doubts about some parts of the test implementation. This result can support teachers and content creators.

5.4 Does prior experience with testing practices influence the student's effectiveness of test case creation and identification? (RQ4)

This research question aims to explore the influence of participants' prior experience with automated testing tools on their ability to identify and create effective test cases on their perspectives. Specifically, it seeks to understand whether participants with prior experience exhibit different levels of effectiveness in test case creation compared to those without prior experience.

The exploration of factors hindering the implementation of test suites is crucial for understanding the challenges students face in software testing education. As highlighted in the study, while automated testing tools like EvoSuite offer promising potential for enhancing testing skills, their adoption may not always translate seamlessly into effective test case implementation. Beyond the technical proficiency in programming languages, students may encounter various obstacles that impede their ability to create robust test suites.

One significant barrier identified in the study is the lack of exposure to comprehensive testing methodologies within the academic curriculum. The survey conducted on computer

science courses across Brazilian federal universities revealed a stark reality: the absence or insufficient coverage of software testing disciplines in a majority of institutions. This deficiency directly correlates with the difficulties students encounter in navigating testing activities. Without structured education in testing principles and techniques, students may struggle to grasp the intricacies of test case identification, implementation, and evaluation.

Moreover, the findings shed light on the nuanced challenges students face even when equipped with foundational programming knowledge. Despite their familiarity with the Java language, participants reported uncertainties and hurdles in implementing test cases effectively. This discrepancy underscores the multifaceted nature of testing proficiency, which extends beyond mere programming skills. The ability to design and execute comprehensive test suites demands a holistic understanding of software systems, requirements analysis, and testing strategies—an understanding that may not be adequately addressed in traditional programming-centric curricula.

In light of these findings, it becomes evident that addressing the deficiencies in software testing education requires a multifaceted approach. Institutions must prioritize the integration of dedicated software testing courses or modules within their curricula, ensuring that students receive comprehensive training in testing methodologies, tools, and best practices. Furthermore, educators should emphasize hands-on experience with automated testing tools like EvoSuite, providing students with practical exposure to industry-standard testing techniques. By bridging the gap between theoretical knowledge and practical application, educational institutions can empower students to navigate the complexities of software testing with confidence and proficiency, thereby enhancing the quality of software development practices in the long run.

6 AI INTEGRATION INTO EDUCATIONAL SOFTWARE TESTING PLATFORM

This chapter presents the second study and its findings, giving continuity to the previous study on software testing education, focusing on the development and evaluation of an innovative software testing learning platform designed to address the unique challenges of teaching this critical skill. Recognizing that effective testing requires both a grasp of complex concepts and the application of practical techniques, this platform aims to provide a comprehensive and engaging learning experience, incorporating Large Language Models (LLMs) like ChatGPT and GPT-4 as potential facilitators in the teaching and learning process. To validate the platform's effectiveness and user-friendliness, we conducted a qualitative study with both computer science students and experienced software testing professionals in northeastern Brazil, whose valuable feedback will be presented and analyzed in detail, providing insights into the platform's functionality and its alignment with the needs and expectations of its target users.

6.1 Context of the software testing teaching

Software testing is one of the pillars of software engineering, playing a critical role in ensuring the quality and reliability of software systems. With society's increasing reliance on computational systems, the demand for qualified professionals in software testing has never been higher. However, effectively teaching this discipline remains a constant challenge for educational institutions worldwide.

Software testing courses represent an essential aspect of programming development education (Aniche *et al.* 2019; Garousi *et al.* 2020; Perkusich 2021). These courses generally adopt a balanced approach between theory and practice, designing test cases to identify flaws in source code (Aniche *et al.* 2019). However, successful learning in this field requires a deep understanding of testing fundamentals coupled with practical skills for effective application (Ferreira *et al.* 2018).

In this context, it is crucial to highlight that students often face substantial challenges when creating comprehensive test suites. Formulating test cases that are both rigorous and representative of real-world usage scenarios is a complex skill that requires practice and guidance (Ferreira *et al.* 2018). Moreover, interpreting results obtained during test execution is also a critical point of interest. Students need to discern between expected results and anomalies that may indicate potential flaws in the code.

Another relevant aspect is the integration of innovative technologies into the software testing teaching process. The use of educational platforms incorporating mutant test analysis, test case generators and AI (artificial intelligence) assistance are a notable example (Zhang *et al.* 2022; Rajabi *et al.* 2023; O'Connor 2023). These advancements allow students to have a more dynamic and interactive learning experience, allowing practical application of acquired knowledge. Furthermore, this approach not only benefits students but also provides valuable insights to teachers regarding individual student progress and areas needing improvement in the teaching process (Beck *et al.* 2000).

This work focuses on enhancing the educational training provided by proposing a platform that combines mutant test analysis with AI assistance via Large Language Models (LLMs) (Liu e Li 2022; Ouh *et al.* 2023) to generate code hints and futurely integrate test case generators. Mutant test analysis, a remarkably effective technique, involves the controlled introduction of "mutants" into the source code, simulating potential faults, with test cases then employed to identify such alterations. Nevertheless, the practical application of this technique can be challenging for students. Introducing pedagogical methods that promote a more solid and practical understanding of this discipline is crucial for shaping qualified professionals prepared for challenges in the Information Technology job market (Edwards 2004; Gopal *et al.* 2021).

Integrating mutant test analysis with test generation signifies a crucial advancement in modernizing software testing education (Valle *et al.* 2015; Paschoal e Souza 2018; Gopal *et al.* 2021). By adopting this approach, students gain a deeper understanding and practical application of cutting-edge technologies prevalent in the software development industry, aligning educational content with industry demands to better prepare future professionals competitively.

A newly introduced tool aimed at assisting in test case creation against mutated code received positive feedback from testers (Valle *et al.* 2015; Paschoal e Souza 2018; Gopal *et al.* 2021). Testers praised the innovative practice feature and the tool's capacity to integrate with personal projects, indicating its effectiveness in teaching core concepts and enhancing practical skills. However, suggestions for enhancing the tool's graphical user interface (GUI) surfaced during interviews, with respondents noting that improvements could render the learning process more accessible and user-friendly, thereby maximizing the tool's efficacy.

Summary: The tool described in this paper is available at the following link: <https://github.com/gabrieldocs/pletes-monorepo>. Detailed instructions for building and running it locally, as well as enabling integration with OpenAI, can be found in the repository.

6.2 Theoretical foundation

This section covers the theoretical framework used to support the study. It discusses the concepts of software testing, mutation testing analysis, the use of artificial intelligence and ChatGPT, as well as the scenario of testing education in Brazil and related works.

6.2.1 Software Testing And Mutation Analysis

Software testing is integral to the development life cycle of applications or systems, ensuring functionality and meeting established requirements (Huizinga e Kolawa 2007; Leppänen *et al.* 2015; Ammann e Offutt 2016; Shahin *et al.* 2017). Various testing approaches, including unit, integration, and acceptance tests, systematically evaluate software to ensure quality, reliability, and error reduction, ultimately delivering more robust products.

Mutation testing analysis evaluates test sets' effectiveness by introducing controlled code mutations and assessing whether existing tests detect them (Coles *et al.* 2016; Ojdanic *et al.* 2023). Identifying uncovered mutations indicates the need to enhance test coverage, emphasizing the technique's significance in code robustness and fragility detection. Overall, software testing serves as a critical precursor to software release, averting post-launch complications, and enabling safe software evolution (Coles *et al.* 2016; Ojdanic *et al.* 2023).

6.2.2 Teaching Testing in Federal Universities in Brazil

In a previous study (Santos *et al.* 2023), analyzing course syllabi from computing courses offered by Brazilian federal universities revealed that among 143 surveyed federal universities, only 30.7% include a dedicated software testing discipline. The remaining universities (69.3%), despite offering computing courses, do not provide a specific software testing discipline (Valle *et al.* 2015; Paschoal e Souza 2018). This gap in the curriculum of many institutions (Gopal *et al.* 2021) can directly impact graduates' preparation to deal with software testing during their professional careers (Edwards 2004).

Developing materials and objects for teaching software testing becomes crucial in shaping more qualified future graduates, impacting the quality assurance of software products in the market (Garousi *et al.* 2020). Tests play an essential role in the development cycle, ensuring defect detection, adherence to requirements, and software robustness (Ammann e Offutt 2016).

Creating materials focused on this domain makes it possible to address a wide range of testing techniques, validation strategies, and available tools in more depth, empowering students with specific and updated knowledge. Moreover, by directing teaching to this area, educational institutions and industry companies contribute to forming a culture of quality, promoting error reduction, increased customer satisfaction, and thus strengthening the competitiveness and reliability of the software industry (Valle *et al.* 2015; Paschoal e Souza 2018; Gopal *et al.* 2021).

6.2.3 Harnessing AI for Enhanced Software Testing and Education

By employing AI-driven test oracles, developers gain crucial insights into the correctness and resilience of their code implementations, thereby fostering a deeper comprehension of testing principles and ultimately promoting more proficient software development practices (Kusuma *et al.* 2022; Ouh *et al.* 2023). In the realm of software testing, the integration of test case generators and oracles for the Java ecossystem, exemplified by tools like EvoSuite (Fraser e Arcuri 2011), Randoop (Pacheco e Ernst 2007) and Toradocu (Goffi *et al.* 2016), has revolutionized testing methodologies. These advanced algorithms automate the generation of test cases and expected outcomes, drastically expediting the testing process while enhancing the quality of code.

Similarly, the emergence of language models (LLMs) has reshaped educational paradigms. LLMs, such as ChatGPT, with its model GPT-4, act as virtual tutors, delivering personalized assistance and real-time feedback to learners (Kusuma *et al.* 2022; Ouh *et al.* 2023). Leveraging their extensive knowledge bases, LLMs aid learners in grasping intricate concepts and refining their skills (Ouh *et al.* 2023). Within software development contexts, LLMs serve as supportive aids for generating code snippets and facilitating test case creation, empowering learners to navigate complex programming tasks more efficiently.

The integration of artificial intelligence (AI) elements, encompassing test case generators, oracles, and language models, presents significant opportunities for enriching education and software development practices (Liu e Li 2022). Furthermore, the broader integration of AI in education holds promise for transforming learning experiences. AI-driven educational

platforms can simulate authentic learning environments, offering hands-on practice in controlled settings. Moreover, AI-powered adaptive learning systems cater to individual student needs, customizing the curriculum to match each learner's pace and proficiency level. This adaptability not only improves learning outcomes but also fosters a more engaging and tailored educational experience.

6.3 Goal and research question

The main goal of this study is to assess the effectiveness of Large Language Models (LLMs) in enhancing the teaching of software testing methodologies. To assess this we proposed the following research question:

***Research Question**: How Can Large Language Models (LLMs), exemplified by GPT-4, be effectively applied in the educational context to improve the teaching of software testing?*

This broad research question allows for an extensive investigation into the potential impact of LLMs in the field of education. It encompasses various aspects such as educational content adaptation, learning personalization, human-machine interaction, knowledge accessibility, and the overall effectiveness of the technology in promoting meaningful learning and academic development among students.

To anchieve this, an innovative approach was adopted, involving the development of an Application Programming Interface (API). This API serves as a pivotal integration tool, facilitating interactions between users and the LLM-powered system. Through this integration, users are provided with a dynamic platform to actively engage in software testing practice sessions, leveraging the intelligent assistance offered by the LLM technology.

6.4 Implementation

In Figure 5, the foundational architecture of our system is depicted, illustrating its core modules and the typical flow of operations. The PLETES application server is central to this architecture, which receives the source code, as shown in the figure. Upon receiving the code, it undergoes versioning and is subsequently submitted to GitHub, facilitating efficient code management and collaboration. Following this, the updated code is synchronized with the application host, where a new Docker Image is constructed, leveraging the power of Docker

technology for containerization and execution.

Once the Docker Image is built, users can initiate the retrieve data task, which triggers the execution of the container. During this step, all relevant information pertaining to the tests generated during the image building process is collected. This data, encompassing test outcomes and other pertinent metrics, is then returned to the user for analysis. Armed with this information, users can iteratively refine their testing strategies, creating and submitting new tests to the platform as necessary. This cyclic process of test creation, execution, and analysis is fundamental to the continuous improvement and optimization of software testing within our system.

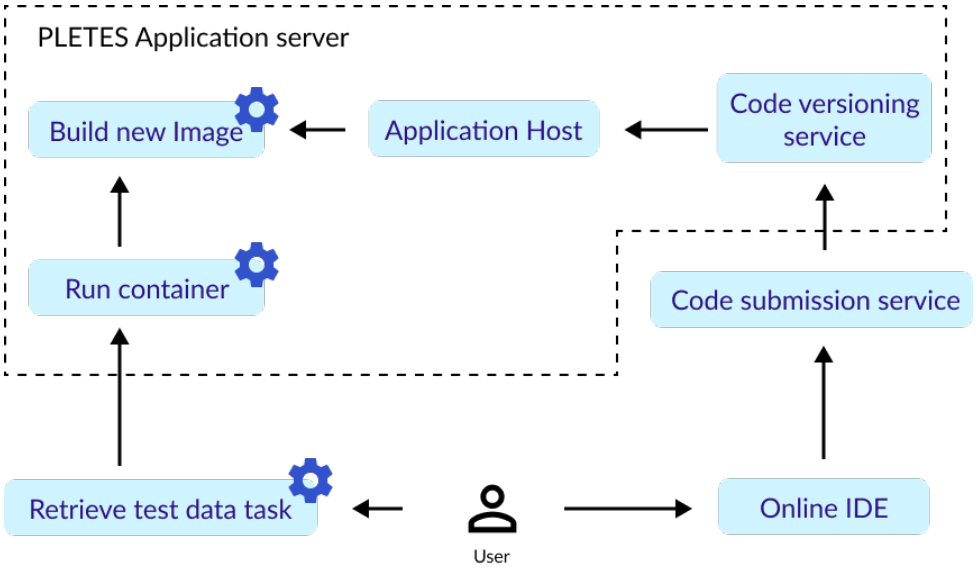

Figure 5 – System diagram of our deployment architecture

Beyond its fundamental functionality, our architecture is designed to accommodate additional components seamlessly. For instance, integration with test case oracles enhances the testing process, augmenting its efficacy. Furthermore, connectivity with advanced LLMs, exemplified by ChatGPT and the model GPT-4, which was adopted in this work, enriches testing capabilities with sophisticated techniques.

The platform offers an integrated development environment (IDE) where users can write and submit their code. This application enable users to perform various operations conveniently. Through the retrieve task users can access important data retrieved from test executions. This includes a summary of how many tests have passed and how many have failed, providing crucial insights into the code's performance. Additionally, tabulated data is returned, with identified mutants along with their status – whether they have been killed by tests or are

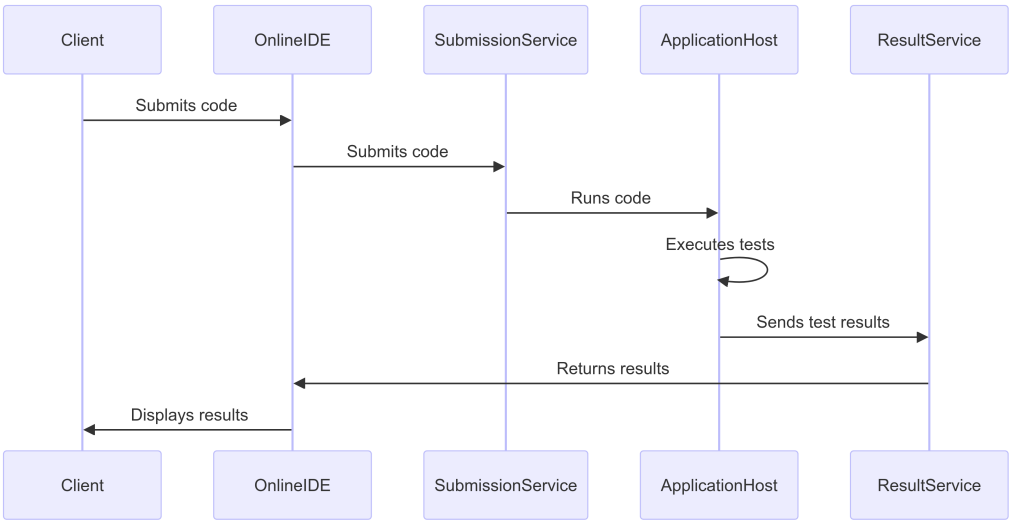

Figure 6 – Simplified Sequence Diagram of Code Submission and Testing Process

still alive. This comprehensive display empowers users to efficiently manage their code and understand its behavior under different conditions.

Our system provides a REST API [1], enabling the development of new applications on its foundation. This accessibility emphasizes openness, as we invite public testing. This transparency encourages collaborative exploration and guarantees the reproducibility of our study's findings. By making our system available to the wider community, our goal is to stimulate innovation and enhance the evolution of software testing methodologies.

In Figure 6, the sequence diagram illustrates the streamlined process of code submission and testing within an online integrated development environment (IDE). The workflow involves five key entities:

1. **Client:** The user who submits the code for testing. 2. **Online IDE:** The web-based platform where code is written and submitted. 3. **Submission Service:** This component receives the code from the IDE, potentially performing initial validation or queueing. 4. **Application Host:** Represents the environment where the submitted code is executed and tested, often a containerized or virtualized setup. 5. **Result Service:** Responsible for collecting, processing, and storing the results of the code execution and tests.

The flow begins with the **Client** submitting their code through the **Online IDE**. The IDE then forwards the code to the **Submission Service**, which handles the initial processing and passes it to the **Application Host** for execution and testing. The **Application Host** then sends the test results to the **Result Service**, which processes and stores them. Finally, the **Result Service** communicates the results back to the **Online IDE**, which displays them to the **Client**,

[1] PLETES, available at: <https://anonymous.4open.science/r/api-pletes-D034/README.md>

completing the feedback loop.

This simplified representation provides a high-level overview of the code submission and testing process, highlighting the key interactions between the involved entities.

6.5 Evaluation

This section presents the responses given by the participants, as well as their perceptions of the tool and approaches used, and opportunities for improvement. As part of our research, we talked with five people. Three of them were master's students who also worked in the industry, and the other two were experienced software quality professionals. We used Google Meet to chat with them online, each conversation lasting between 40 minutes to 1 hour and 15 minutes. We talked about our platform and how users use it. After each chat, we wrote down everything they said so we could study it later and learn from their experiences.

6.5.1 Interview Responses

The participants' responses offer valuable insights into expectations and perceptions regarding the educational platform. One highlighted aspect is the appreciation for the feedback system, seen as a way to reward students' efforts. This strategy can be particularly effective in encouraging student engagement with the content, providing a sense of progress, and reinforcing the importance of continuous learning.

There was also a critical observation regarding feedback, suggesting the possibility of including more comprehensive responses, such as displaying the JUnit and Pitest output terminals. This suggestion indicates a desire for greater detail and clarity in assessments, enriching the users' learning experience.

The suggestion to provide a mobile interface is a relevant observation. By offering portability, the platform becomes more accessible and flexible, allowing students to access content conveniently regardless of the device used. This improvement would enhance the platform's reach and cater to a range of user profiles.

Overall, participants' responses provide valuable insights to enhance the educational platform. Positive feedback on the feedback system and the intuitive interface highlights strong points to be maintained, while improvement suggestions indicate areas of opportunity to optimize the learning experience and make the platform even more effective and accessible.

6.5.2 Perception Of User Interface And Usage Flow

Perceptions of the platform's interface and usage flow present a mix of positive impressions and improvement suggestions. One interviewee, a professional in the software testing field, expressed appreciation for the interface's simplicity, highlighting step numbering and the inclusion of the LLMs (Schäfer *et al.* 2024) as notable elements. When commenting on the platform's mobile accessibility, he notes that *"it works on mobile devices too, as it is hosted on the web"*. However, he also points out an area for improvement, indicating that responsiveness could be optimized. Figure 7 displays an example of code using a Java class. The left side panel displays different classes users can interact. The center of the figure displays the text editor. The bottom displays the prompt where user can interact with the AI assistant.

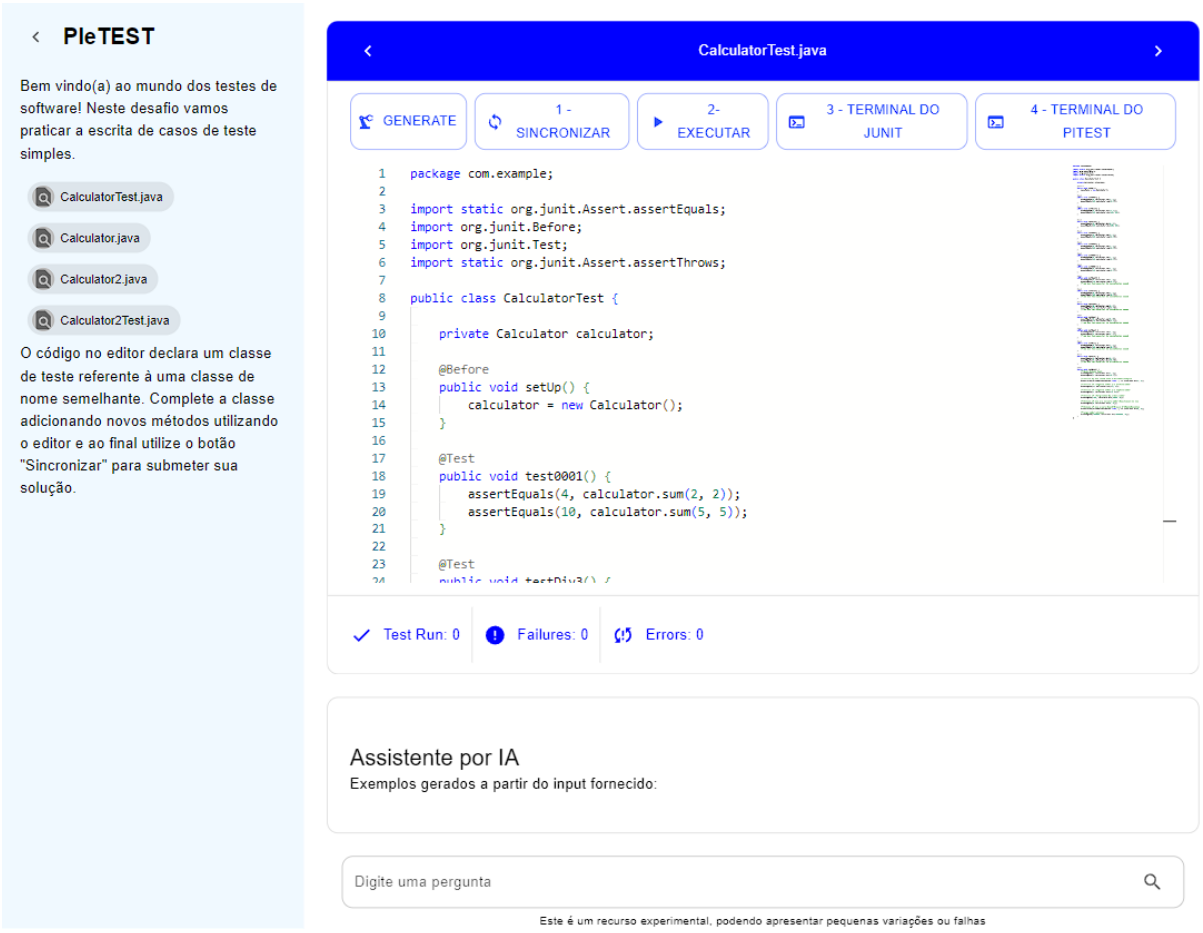

Figure 7 – Screenshot of PLETES web client interface.

The interviewee also emphasizes the importance of intuitive and informative documentation, especially for beginners in the software testing field. He expresses concern about the usability of testing tools, mentioning that he has encountered situations where *"the tool was*

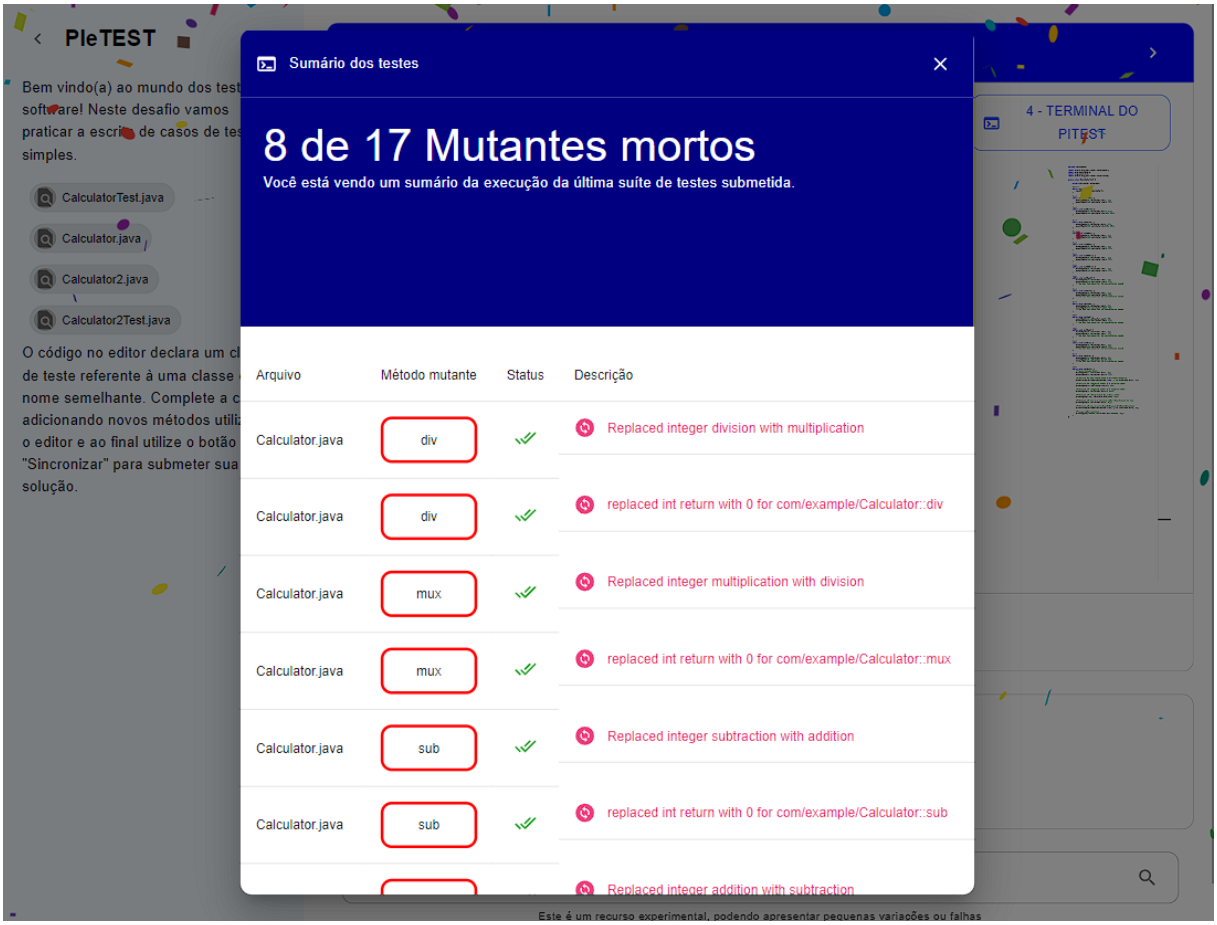

Figure 8 – Screenshot of PLETES score screen.

very complete, but the user didn't know it was complete because it didn't make clear what it could do". In this regard, he highlights the relevance of elements that assist in understanding and effectively using the platform.

Another interviewee, a postgraduate student, showed a positive view of the testing education tool's interface. He praised the arrangement of elements on the platform, stating that "the buttons are very suggestive of what to do." Additionally, he highlighted the usefulness of functional shortcuts in the editor, favorably comparing it to VISUAL STUDIO CODE. However, he mentioned slight confusion about the synchronization function, suggesting that this aspect could be clarified.

Regarding AI assistance in code generation and test suggestions, the interviewee expressed enthusiasm. He saw great potential in the AI's ability to provide additional scenarios for creating new test cases, making tests more comprehensive and complete, and clarifying any doubts about the source code.

In summary, participants appreciated the tool's interface, praising its practicality and the integration of the editor with AI assistance. They also saw significant benefits in the tool's

approach to generating more comprehensive tests. These positive observations indicate that the platform has elements that can facilitate the software testing learning process. However, some suggestions were raised regarding the source code synchronization function, indicating that there is still room for improvement in the user experience.

In conclusion, the perception of the platform's interface is generally positive, emphasizing simplicity and including features such as step numbering and AI assistance. However, the interviewee identifies specific areas that can be improved, such as responsiveness on mobile devices and the need for informative documentation to ensure a more effective user experience, especially for those taking their first steps in the field of software testing.

6.5.3 Perception Of The AI Integration

The LLMs proved to be an extremely valuable tool, as reported by the participants. One highlighted point was its ability to provide assistance in situations where users face testing challenges or have doubts about accessing specific code portions. This functionality proved particularly useful in resolving deadlocks and providing guidance in complex development scenarios. Furthermore, the LLMs proved to be an effective source of inspiration for creating new test cases and exploring unknown flows. Participants perceived that the AI had the ability to stimulate creativity and promote a more comprehensive approach to test design, crucial for ensuring complete and effective coverage, as reported by one participant:

> *"[...] I've had contact with ChatGPT, and in my use, it helped identify unmapped situations, more elaborate verification scenarios, and approach suggestions that brought excellent results in my use. I believe it would be the same experience with PleTes."*

The participant highlights their positive experience using ChatGPT, emphasizing that the tool was effective in identifying previously unconsidered situations and in elaborating more complex verification scenarios. Additionally, another participant emphasizes that the approach suggestions provided by ChatGPT resulted in very satisfactory outcomes. By expressing the belief that they would have a similar experience with PleTes, the participant indicates an optimistic expectation regarding the utility and effectiveness of this new platform.

> *"This platform offers an innovative approach to tackling challenges in programming learning. By providing precise guidance on complex code snippets, either directly*

or through AI assistance, it helps students overcome obstacles and progress in their learning process. This is especially valuable to prevent frustration and dropout, providing a more effective and rewarding learning experience."

The impact of integrating ChatGPT and similar tools into the platform was also noteworthy. The interviewee expressed surprise at discovering the broad capabilities that this integration enabled. The ease with which the LLMs was incorporated into the platform and its significant contribution to enhancing the efficiency and effectiveness of testing processes proved to be a surprising revelation for users.

Another postgraduate student mentioned their curiosity about AI-based text and code generation assistants, such as GPT, admitting to interacting with them out of pure curiosity. However, they did not request that the AI perform code refactoring or implement specific code.

Regarding the integration of AI into the platform, they mentioned: *"So, even if the output is not exactly what it is, it can help us create new scenarios to make testing more complete. So, this type of tool is very interesting."* These quotes highlight the importance of practice for effective learning and the potential of AI to assist in generating more comprehensive test scenarios.

In summary, the LLMs stood out as a highly valuable resource for participants. Its ability to guide in challenging situations, stimulate creativity in test design, and surprise users with its integrated capabilities on the platform demonstrated its potential to significantly transform and enhance testing processes.

Type of Interaction	**Programming Use Cases**	**Test Creation**
Development Assistance	Assists in code challenges Provides guidance for implementation	Suggests test cases Stimulates creativity in design
Code Generation	Creates code snippets Optimizes and automates processes	Generates test scripts Facilitates creation of automated tests
Static Analysis	Identifies syntax errors Suggests code improvements	Identifies coverage gaps Optimizes test coverage

Table 3 – Forms of Interaction with platform assistance and Benefits for Programming and Testing

Table 3 outlines some of the possibilities raised by participants regarding various forms of interaction with AI and the associated benefits for programming and test creation. The following relates these elements to the responses provided by participants. Relating this to

participants' responses, we observe congruence between user perceptions and the capabilities identified. This highlights the utility and effectiveness of AI as a support tool in the programming and test creation processes.

6.5.4 Research Question: How Can Large Language Models (LLMs), Exemplified by GPT-4, Be Effectively Applied in the Educational Context to Improve the Teaching of Software Testing?

Large Language Models (LLMs), such as the GPT-4 model, have arised as a valuable tool in the software engineering educational scenario. LLMs play a important role facilitating comprehension and student engagement across a wide range of topics and diverse educational levels. Insights gathered from the study's participants underscore the profound impact of LLMs in overcoming educational challenges and facilitating test learning. Notably, these models are mentioned for their ability to inspire creativity and streamline the process of test design. For instance, ChatGPT, received praise for its adeptness in identifying unexplored scenarios and suggesting verification methods, thereby yielding favorable results. Moreover, participants expressed optimism regarding the potential of similar tools like PLETES, indicating a broader recognition of the transformative potential of AI-driven platforms in education.

Moreover, the integration of LLMs into educational platforms has yielded surprising benefits, significantly transforming testing processes and enhancing efficiency. Users were pleasantly surprised by the broad capabilities enabled by this integration, which facilitated the creation of new and more comprehensive test scenarios. The ability of LLMs to provide guidance on complex topics and assist in generating test scripts underscores their potential to prevent frustration and dropout among students, ultimately offering a more effective and rewarding learning experience. Overall, the integration of LLMs exemplified by GPT-4 holds promise in revolutionizing educational practices, facilitating access to knowledge, and fostering interactive learning environments across various disciplines and educational levels.

7 DISCUSSIONS

This work presents the execution of a software testing experiment aimed at assessing students' perceptions of test case creation in computer-related courses. Participants were tasked with identifying and implementing test cases from a provided software system. The study involved two groups: a control group provided with the source code and documentation, and an experimental group, which, in addition to the source code and documentation, received a test suite generated by the EvoSuite tool. Participants completed three questionnaires: one regarding the project, another assessing their prior knowledge profile, and a third providing feedback on their perception of the experiment and software testing education.

Results indicated that the EvoSuite tool facilitated the discovery of new test scenarios and enabled the identification of novel approaches to testing code segments previously unfamiliar to participants. Overall, the findings suggest that integrating code generation tools like EvoSuite into software testing education can enhance students' testing skills and improve test quality.

Furthermore, this research highlights emerging trends in software testing education, such as the increasing emphasis on practical, hands-on learning experiences. Traditional lecture-based approaches are being supplemented by interactive workshops, simulations, and real-world case studies to foster a deeper understanding of testing principles and techniques. Another key trend is the integration of automation into software testing curricula, equipping students with proficiency in automated testing tools. Additionally, there is a growing recognition of the importance of soft skills in software testing education, such as effective communication, teamwork, critical thinking, and problem-solving abilities.

The study also underscores the significance of unit testing and test automation. Unit tests evaluate the smallest units of code, while automated tests are essential for continuous integration practices. The adoption of test automation tools in educational programs can provide students with critical skills needed for modern software development environments.

The introduction of the EVOSUITE tool in the classroom provides students with the opportunity to work with more extensive and complex code, something that many of them would not have such accessible exposure to in a traditional academic environment. The automated test generation with EvoSuite allows students to explore more substantial projects, encompassing a wider variety of functionalities and challenges that may be encountered in real-world systems. This hands-on experience with larger codebases is valuable for students' education as it prepares them to face more realistic situations they may encounter in their professional careers.

For those students who are not familiar with test creation, using the test suite generated by EVOSUITE is an excellent learning opportunity. They can observe how EVOSUITE identifies and automatically constructs a series of test cases that address different parts of the code. Furthermore, by analyzing the test coverage obtained with this suite, students can better understand the effectiveness of the tests performed and identify areas of the code that may not be adequately tested. This critical analysis helps them develop a deeper understanding of the importance of test coverage and how it can be enhanced to ensure the quality of the developed software.

Therefore, the use of the EVOSUITE tool in the classroom not only allows students to work with larger codebases but also offers a practical and effective approach for those less familiar with test creation. With this powerful tool, students have the chance to enhance their skills in software testing and acquire essential knowledge to become competent professionals ready to tackle real-world challenges in software development. Furthermore, the potential benefits of automated test generation extend beyond specific programming languages or development environments. By comparing and analyzing the tool's performance in different contexts, valuable insights will be gained into its general applicability and limitations. Educators can then use this knowledge to adapt their instructional approaches and curricula effectively to specific academic contexts.

As a result of this research, it became evident that the EVOSUITE tool facilitated the discovery of new test scenarios and offered new ways to test code snippets with which students were previously unfamiliar. By integrating such tools into the teaching process, students were able to improve their understanding and application of software testing concepts.

This research holds implications for educators and students in computer science-related courses, offering insights into effective approaches for enhancing software testing education by embracing hands-on learning, and fostering the development of soft skills.

8 THREATS TO VALIDITY

This chapter aims to present the threats to validity encountered during the conception of the study (W. 2014). Section 8.1 presents threats to construct validity. Then Section 8.2 addresses threats to internal validity of the work. Section 8.3 outlines threats to external validity of the work, and finally, Section 8.3 discusses threats to conclusion validity of the work.

8.1 Contruction validation

The conduct of a pilot stud for the experiment in the first study was of paramount importance to ensure the quality and effectiveness of the work, as well as to minimize potential points of difficulty that could arise during the main experiment. This pilot study consisted of a group of three participants who underwent internal validation of the experiment and its artifacts. The main objective was to identify and correct potential problems before the application phase.

During the pilot study, all proposed activities were carefully evaluated, and only those that could be completed within the planned duration for the experiment were retained, thus avoiding situations that could hinder the analysis of results. Additionally, the environment selected for the experiment application was chosen to be easily adjustable, allowing for quick modifications and adaptations if improvement needs were identified.

This preliminary validation process allowed the identification of flaws or limitations in procedures, ensuring that the main experiment was conducted with the highest possible efficiency and accuracy. Based on the results of the pilot study, relevant adjustments were made to the experiment parameters to optimize participants' experience and ensure more robust and reliable data collection.

Thus, the pilot study played a fundamental role in the preparation and refinement of the experiment, providing a solid foundation for its successful execution. Through this preliminary stage, it was ensured that the work was adequately structured, avoiding potential obstacles and increasing the validity of the achieved results.

8.2 Internal validation

Furthermore, aiming to minimize comprehension difficulties with the tasks proposed in the experiment or any potential issues with the artifacts used, several strategies were adopted. One of the main measures was conducting a review session before the experiment began, during

which key topics related to the tests to be performed were revisited, along with practical examples of accessing certain code segments. The purpose of this session was to provide a solid knowledge base and clarify any doubts participants might have, ensuring everyone was familiar with the tools and procedures involved in executing the experiment.

Throughout the experiment, participants had access to continuous assistance, allowing them to seek help for more general issues related to the experimental environment and task execution. The assistance team was available to impartially answer questions, without influencing or directing participants' responses. This approach ensured that all involved could approach the experiment with confidence and clarity, guaranteeing the quality and accuracy of the results obtained.

By combining strategies such as the review session and assistance during the experiment, the aim was to provide an environment conducive to participants effectively engaging with the proposed tasks, fully understanding the study's objectives and procedures. These measures were essential to ensure the reliability of the results and to ensure that any undesirable effects resulting from potential comprehension difficulties were minimized, thus consolidating the validity of the research conducted.

8.3 External validation

This experiment was conducted on a group of 35 students (undergraduate and graduate students) who, despite belonging to various semesters and having different levels of knowledge regarding prior knowledge and professional experiences, may not represent a significant sample of Brazilian students. To mitigate these effects, participants were randomly distributed into each experimental group. The experiment was planned to use the Java language, as it is the language that students are most familiar with and use most frequently in other disciplines in the course.

It is important to note that this study provides a basis for future work and extensions. To broaden the generalization of the results, the study can be replicated in other programming languages such as Python, JavaScript, Asp.NET, among others. Additionally, considering different test generation tools may help to better understand how this approach performs in other contexts.

EvoSuite was chosen for this scenario because it offers the advantage of not requiring major changes to the existing software project for test cases to be generated and executed. On

the other hand, to use Randoop or Toradocu instead of EvoSuite, some additional adaptations to the software project would be necessary. Randoop is another test generation tool that uses the technique of random test case generation, which may require more adjustments and configurations in the code for it to work effectively. Toradocu, on the other hand, is a tool specifically for contract test generation from specifications in Javadoc format, requiring the source code to contain these annotations for Toradocu to generate tests according to the specifications.

8.4 Conclusion validation

To ensure an accurate and reliable analysis of the collected data, several measures were adopted to mitigate potential misunderstandings or distortions in the results. One of the key strategies used was to conduct rigorous data treatments, aiming to enable a more assertive analysis throughout various stages of the study. In this regard, techniques for data filtering and tabulation were employed, allowing them to be consistently applied across all phases of the research.

To ensure the quality of the data used, any records considered incomplete were removed to prevent missing information from affecting the interpretation of the results. Additionally, outliers, extreme and atypical data points that could distort the analysis, were also identified and excluded from the dataset.

9 CONCLUSIONS

The present study evaluated, through an experiment, the perception of students when dealing with software testing activities. It was observed that the EVOSUITE tool can be applied to support the teaching and learning process, but further studies and tests are needed to monitor the use of this tool throughout a course. Among the groups that can benefit from the results of this work are teachers and students of computer science courses who are studying testing practices or planning to start studying software testing. It also concluded that the development and adoption of tools like PLETES could benefit students and teachers to practice their skills on software testing while the integration of artificial intelligence elements such as large language models (LLMs) could help students to quickly debug and discuss ideas on their assignments.

9.1 Conclusions on the EVOSUITE adoption for testing teaching

The first study proposes the integration of automated test generation tools, such as EVOSUITE in the software testing education. It highlights the potential benefits of the tool and advocates for further exploration and adaptation to optimize its impact on students' learning experiences. Several articles highlight the challenges in creating suitable materials for teaching software testing. Alongside the second study we address this issue by providing a methodology for instructors. This methodology includes test oracles for generating test suites and mutation testing analysis to automate the evaluation process and measure the quality of students' work. By offering these tools and techniques, we aim to empower educators to create more effective and engaging learning experiences in software testing. The automated assessment capabilities not only save instructors valuable time but also provide students with immediate feedback on their progress, fostering a deeper understanding of testing principles and practices.

The introduction of the EVOSUITE tool in the classroom has provided students with the opportunity to work with more extensive and complex codebases, something that is often limited in traditional academic environments. This exposure is invaluable as it mirrors the challenges and scenarios students are likely to encounter in their professional careers. The ability to work with larger projects and diverse functionalities enhances their practical understanding and prepares them for real-world software development tasks.

For students who lack experience in test creation, the automated test generation offered by EVOSUITE presents an excellent learning opportunity. By observing how the tool

constructs test cases and analyzing the resulting test coverage, students gain insights into effective testing strategies and the importance of thorough test coverage. This experience not only improves their technical skills but also instills a deeper appreciation for the critical role of testing in software quality assurance.

Moreover, the hands-on experience with automated testing tools addresses a significant gap in current educational curricula, which often emphasize programming skills over comprehensive testing methodologies. By integrating such tools, educational programs can provide a more balanced and holistic software engineering education. The practical experience gained from using these tools helps students develop a more robust skill set that includes both coding and testing competencies, thereby enhancing their overall readiness for the software industry.

The study also underscores the need for continuous evaluation and adaptation of educational tools to meet the evolving demands of software development. While the initial findings are promising, ongoing research and iterative improvements are essential to maximize the educational benefits of tools like EvoSuite. This includes tailoring the tool's usage to different academic contexts and levels of student proficiency. Ultimately, the integration of automated test generation tools into the curriculum represents a significant step forward in modernizing software testing education. It equips students with essential skills and knowledge, prepares them for industry challenges, and enhances the overall quality of software engineering education. As educational institutions continue to embrace these advancements, they can produce more competent, confident, and industry-ready software professionals.

9.2 Conclusions on the PLETES platform

Students have multifaceted expectations for the PLETES platform, ranging from desiring a more dynamic and engaging learning experience to aiming for practical skill enhancement in software testing. They view the incorporation of playful elements as motivational tools for grasping complex concepts effectively and see interaction with Learning Language Models (LLMs) as valuable for practicing test case formulation and clarifying doubts.

Moreover, students anticipate personalized learning experiences through PLETES, seeking to tailor their exploration of software testing concepts to align with their professional aspirations. They expect the platform to offer a comprehensive overview of various testing types, empowering them to navigate diverse testing scenarios and acquire skills applicable beyond the

platform to real-world software development.

Despite its novelty and utility, the PLETES platform faces limitations that warrant consideration. These include potential constraints in simulating a diverse array of testing scenarios, which might hinder students' exploration of nuanced software testing complexities. Pedagogical barriers such as adapting to the virtual learning environment and utilizing the platform's features effectively could also pose challenges, particularly for students unfamiliar with such educational tools. Ensuring platform accessibility across varying technical proficiency levels is imperative, necessitating proactive measures to support students encountering initial difficulties. Thus, addressing these limitations is vital for optimizing the PLETES platform's educational efficacy and ensuring equitable access for all students.

9.3 Future research

A potential line of research concerns the expansion and adaptation of tools like the EVOSUITE and PLETES platform for different contexts and target audiences. Investigating the effectiveness of the platform in various scenarios, such as technical courses or distance learning environments, can provide a more comprehensive understanding of how the playful approach can be optimized to meet the specific educational needs of different student groups.

Incorporating a catalog of common code faults would significantly enhance the user experience by providing a centralized resource for reference and assistance. This catalog could serve as a knowledge base, offering detailed explanations of typical errors, their underlying causes, and potential solutions. By categorizing and indexing these faults, users could quickly identify relevant information, saving valuable time and effort during the debugging process. Furthermore, the catalog could be expanded to include real-world examples, best practices, and links to external resources, creating a comprehensive toolkit for addressing coding challenges. This would not only accelerate the troubleshooting process but also promote learning and skill development among developers.

Furthermore, the integration of performance metrics and evaluation within the platform could enhance the ability to monitor student progress more accurately. This would allow educators to tailor content and activities based on individual performance, further personalizing the learning experience.

BIBLIOGRAPHY

AMMANN, P.; OFFUTT, J. **Introduction to Software Testing**. 2. ed. [S.l.]: Cambridge University Press, 2016.

ANICHE, M.; HERMANS, F.; DEURSEN, A. van. Pragmatic software testing education. In: **Proceedings of the 50th ACM Technical Symposium on Computer Science Education**. [s.n.], 2019. ISBN 9781450358903. Disponível em: <https://doi.org/10.1145/3287324.3287461>.

BASILI, V. R.; ROMBACH, H. D. The tame project: Towards improvement-oriented software environments. **IEEE Trans. Software Eng.**, v. 14, p. 758–773, 1988. Disponível em: <https://api.semanticscholar.org/CorpusID:41577460>.

BECK, J.; WOOLF, B. P.; BEAL, C. R. Advisor: A machine learning architecture for intelligent tutor construction. In: **Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence**. [S.l.]: AAAI Press, 2000. p. 552–557. ISBN 0262511126.

BENITTI, F. B. V. A methodology to define learning objects granularity: A case study in software testing. **Informatics in Education**, Vilnius University Institute of Data Science and Digital Technologies, v. 17, n. 1, p. 1–20, 2018. ISSN 1648-5831.

BIJLSMA, L.; DOORN, N.; PASSIER, H.; POOTJES, H.; STUURMAN, S. How do students test software units? In: **Proceedings of the 43rd International Conference on Software Engineering: Joint Track on Software Engineering Education and Training**. [s.n.], 2021. ISBN 9780738133201. Disponível em: <https://doi.org/10.1109/ICSE-SEET52601.2021.00029>.

COLES, H.; LAURENT, T.; HENARD, C.; PAPADAKIS, M.; VENTRESQUE, A. Pit: A practical mutation testing tool for java (demo). In: **Proceedings of the 25th International Symposium on Software Testing and Analysis**. New York, NY, USA: Association for Computing Machinery, 2016. (ISSTA 2016), p. 449–452. ISBN 9781450343909. Disponível em: <https://doi.org/10.1145/2931037.2948707>.

CORTE, C. K. D.; RIESTIN, A. C.; SILVA, M. A. G.; BARBOSA, E. F.; MALDONADO, J. C. Progtest: Ambiente para submissão e avaliação de trabalhos práticos. In: **XVIII Simpósio Brasileiro de Informática na Educação (SBIE 2007) Workshop sobre Ambientes de Apoio à Aprendizagem de Algoritmos e Programação**. São Paulo, SP: [s.n.], 2007.

EDWARDS, S. H. Using software testing to move students from trial-and-error to reflection-in-action. **SIGCSE Bull.**, Association for Computing Machinery, New York, NY, USA, v. 36, n. 1, p. 26–30, mar 2004. ISSN 0097-8418. Disponível em: <https://doi.org/10.1145/1028174.971312>.

FERREIRA, F.; VALE, G.; DINIZ, J. P.; FIGUEIREDO, E. On the proposal and evaluation of a test-enriched dataset for configurable systems. In: **Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems**. New York, NY, USA: Association for Computing Machinery, 2020. (VaMoS '20). ISBN 9781450375016. Disponível em: <https://doi.org/10.1145/3377024.3377045>.

FERREIRA, T.; VIANA, D.; FERNANDES, J.; SANTOS, R. Identifying emerging topics and difficulties in software engineering education in brazil. In: **Proceedings of the XXXII Brazilian Symposium on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2018. ISBN 9781450365031. Disponível em: <https://doi.org/10.1145/3266237.3266247>.

FRASER, G.; ARCURI, A. Evosuite: Automatic test suite generation for object-oriented software. In: **Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2011. (ESEC/FSE '11), p. 416–419. ISBN 9781450304436. Disponível em: <https://doi.org/10.1145/2025113.2025179>.

GAROUSI, V.; FELDERER, M.; KUHRMANN, M.; HERKILOgLU, K. What industry wants from academia in software testing? hearing practitioners' opinions. In: **Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2017. (EASE'17), p. 65–69. ISBN 9781450348041. Disponível em: <https://doi.org/10.1145/3084226.3084264>.

GAROUSI, V.; RAINER, A.; LAUVåS, P.; ARCURI, A. Software-testing education: A systematic literature mapping. **Journal of Systems and Software**, v. 165, p. 110570, 2020. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121220300510>.

GOFFI, A.; GORLA, A.; ERNST, M. D.; PEZZè, M. Automatic generation of oracles for exceptional behaviors. In: **Proceedings of the 25th International Symposium on Software Testing and Analysis**. New York, NY, USA: Association for Computing Machinery, 2016. (ISSTA 2016), p. 213–224. ISBN 9781450343909. Disponível em: <https://doi.org/10.1145/2931037.2931061>.

GOPAL, B.; COOPER, S.; OLMANSON, J.; BOCKMON, R. Student difficulties in unit testing, integration testing, and continuous integration: An exploratory pilot qualitative study. **Psychology of Programming Interest Group**, 2021.

HAZZAN, O.; HAR-SHAI, G. Teaching computer science soft skills as soft concepts. In: **Proceeding of the 44th ACM Technical Symposium on Computer Science Education**. New York, NY, USA: Association for Computing Machinery, 2013. (SIGCSE '13), p. 59–64. ISBN 9781450318686. Disponível em: <https://doi.org/10.1145/2445196.2445219>.

HUIZINGA, D.; KOLAWA, A. **Automated Defect Prevention: Best Practices in Software Management**. [S.l.: s.n.], 2007. ISBN 0470042125.

JOSHI, A.; KALE, S.; CHANDEL, S.; PAL, D. K. Likert scale: Explored and explained. **British Journal of Applied Science and Technology**, v. 7, p. 396–403, 2015. Disponível em: <https://api.semanticscholar.org/CorpusID:49525257>.

JR, E. O.; COLANZI, T.; AMARAL, A.; CORDEIRO, A.; NETO, J. C.; SOUZA, S. Ensino, aprendizagem e uso profissional da uml em maringá e região. In: **Anais do XXIX Workshop sobre Educação em Computação**. Porto Alegre, RS, Brasil: SBC, 2021. p. 328–337. ISSN 2595-6175. Disponível em: <https://sol.sbc.org.br/index.php/wei/article/view/15924>.

KUSUMA, J. S.; HALIM, K.; PRANOTO, E. J. P.; KANIGORO, B.; IRWANSYAH, E. Automated essay scoring using machine learning. In: **2022 4th International Conference on Cybernetics and Intelligent System (ICORIS)**. [S.l.: s.n.], 2022. p. 1–5.

LEPPäNEN, M.; MäKINEN, S.; PAGELS, M.; ELORANTA, V.-P.; ITKONEN, J.; MäNTYLä, M. V.; MäNNISTö, T. The highways and country roads to continuous deployment. **IEEE Software**, v. 32, n. 2, p. 64–72, Mar 2015. ISSN 1937-4194.

LIU, X.; LI, Y. Redefining teacher qualification in the artificial intelligence era: A professional capital perspective. In: **Proceedings of the 5th International Conference on Big Data and Education**. New York, NY, USA: Association for Computing Machinery, 2022. (ICBDE '22), p. 35–39. ISBN 9781450395793. Disponível em: <https://doi.org/10.1145/3524383.3524405>.

LOJO, N.; FOX, A. Teaching test-writing as a variably-scaffolded programming pattern. In: **Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1**. New York, NY, USA: Association for Computing Machinery, 2022. (ITiCSE '22), p. 498–504. ISBN 9781450392013. Disponível em: <https://doi.org/10.1145/3502718.3524789>.

MENDES, J.; COSTA, Y.; FRAZãO, K.; SANTOS, R.; SANTOS, D.; RIVERO, L. Identificação das expectativas e dificuldades de alunos de graduação no ensino de engenharia de software. In: **Anais do XXVII Workshop sobre Educação em Computação**. Porto Alegre, RS, Brasil: SBC, 2019. p. 334–347. ISSN 2595-6175. Disponível em: <https://sol.sbc.org.br/index.php/wei/article/view/6640>.

MOREIRA, G. G.; REINEHR, S.; MALUCELLI, A.; AMSTEL, F. V. Protesters: A board game for teaching the testing process. In: **Proceedings of the XXI Brazilian Symposium on Software Quality**. New York, NY, USA: Association for Computing Machinery, 2023. (SBQS '22). ISBN 9781450399999. Disponível em: <https://doi.org/10.1145/3571473.3571503>.

OJDANIC, M.; SOREMEKUN, E.; DEGIOVANNI, R.; PAPADAKIS, M.; TRAON, Y. L. Mutation testing in evolving systems: Studying the relevance of mutants to code evolution. **ACM Trans. Softw. Eng. Methodol.**, Association for Computing Machinery, New York, NY, USA, v. 32, n. 1, feb 2023. ISSN 1049-331X. Disponível em: <https://doi.org/10.1145/3530786>.

OUH, E. L.; GAN, B. K. S.; SHIM, K. J.; WLODKOWSKI, S. Chatgpt, can you generate solutions for my coding exercises? an evaluation on its effectiveness in an undergraduate java programming course. In: **Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1**. New York, NY, USA: Association for Computing Machinery, 2023. (ITiCSE 2023), p. 54–60. ISBN 9798400701382. Disponível em: <https://doi.org/10.1145/3587102.3588794>.

O'CONNOR, S. Corrigendum to "open artificial intelligence platforms in nursing education: Tools for academic progress or abuse?" [nurse educ. pract. 66 (2023) 103537]. **Nurse Education in Practice**, v. 67, p. 103572, 2023. ISSN 1471-5953. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1471595323000343>.

PACHECO, C.; ERNST, M. D. Randoop: feedback-directed random testing for java. In: **Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion**. New York, NY, USA: Association for Computing Machinery, 2007. (OOPSLA '07), p. 815–816. ISBN 9781595938657. Disponível em: <https://doi.org/10.1145/1297846.1297902>.

PASCHOAL, L. N.; SOUZA, S. de. A survey on software testing education in brazil. In: **Anais do XVII Simpósio Brasileiro de Qualidade de Software**. Porto Alegre, RS, Brasil: SBC, 2018. p. 334–343. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/sbqs/article/view/15088>.

PERKUSICH, M. E. S. e Tamires Rocha e M. O ensino de engenharia de software no nível superior: um mapeamento sistemático. **Revista Principia - Divulgação Científica e Tecnológica**

do IFPB, v. 1, n. 56, p. 116–125, 2021. ISSN 2447-9187. Disponível em: <https://periodicos.ifpb.edu.br/index.php/principia/article/view/3874>.

PETROVIć, G.; IVANKOVIć, M.; FRASER, G.; JUST, R. Practical mutation testing at scale: A view from google. **IEEE Transactions on Software Engineering**, v. 48, n. 10, p. 3900–3912, 2022.

RAJABI, P.; TAGHIPOUR, P.; CUKIERMAN, D.; DOLECK, T. Exploring chatgpt's impact on post-secondary education: A qualitative study. In: **Proceedings of the 25th Western Canadian Conference on Computing Education**. New York, NY, USA: Association for Computing Machinery, 2023. (WCCCE '23). ISBN 9798400707896. Disponível em: <https://doi.org/10.1145/3593342.3593360>.

SANTOS, L.; SOUSA, N.; GOMES, L.; GARCIA, J.; FERREIRA, F. An experiment on software test creation: can evosuite help test teaching? In: **Proceedings of the XXII Brazilian Symposium on Software Quality**. New York, NY, USA: Association for Computing Machinery, 2023. (SBQS '23), p. 281–290. ISBN 9798400707865. Disponível em: <https://doi.org/10.1145/3629479.3629499>.

SCHäFER, M.; NADI, S.; EGHBALI, A.; TIP, F. An empirical evaluation of using large language models for automated unit test generation. **IEEE Transactions on Software Engineering**, v. 50, n. 1, p. 85–105, Jan 2024. ISSN 1939-3520.

SETIANI, N.; FERDIANA, R.; SANTOSA, P. I.; HARTANTO, R. Literature review on test case generation approach. In: **Proceedings of the 2nd International Conference on Software Engineering and Information Management**. New York, NY, USA: Association for Computing Machinery, 2019. (ICSIM 2019), p. 91–95. ISBN 9781450366427. Disponível em: <https://doi.org/10.1145/3305160.3305186>.

SHAHIN, M.; BABAR, M. A.; ZHU, L. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. **IEEE Access**, v. 5, p. 3909–3943, 2017. ISSN 2169-3536.

SOUZA, D.; MALDONADO, J. C.; BARBOSA, E. F. Aspectos de desenvolvimento e evolução de um ambiente de apoio ao ensino de programação e teste de software. **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)**, v. 23, n. 1, 2012. ISSN 2316-6533. Disponível em: <http://ojs.sector3.com.br/index.php/sbie/article/view/1695>.

SáNCHEZ, A. B.; DELGADO-PéREZ, P.; MEDINA-BULO, I.; SEGURA, S. Mutation testing in the wild: Findings from github. **Empirical Softw. Engg.**, Kluwer Academic Publishers, USA, v. 27, n. 6, nov 2022. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-022-10177-8>.

VALLE, P. H. D.; BARBOSA, E. F.; MALDONADO, J. C. Cs curricula of the most relevant universities in brazil and abroad: Perspective of software testing education. In: **2015 International Symposium on Computers in Education (SIIE)**. [S.l.: s.n.], 2015. p. 62–68.

VOGL, S.; SCHWEIKL, S.; FRASER, G.; ARCURI, A.; CAMPOS, J.; PANICHELLA, A. Evosuite at the sbst 2021 tool competition. In: IEEE. **2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)**. [S.l.], 2021. p. 28–29.

W., C. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In: **Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)**. [S.l.: s.n.], 2014. p. 1–10.

ZANETTI, H. A. P.; BORGES, M. A. F.; RICARTE, I. L. M. Comfapoo: Método de ensino de programação orientada à objetos baseado em aprendizagem significativa e computação física. **Revista Brasileira de Informática na Educação**, v. 31, p. 01–30, jan. 2023. Disponível em: <https://sol.sbc.org.br/journals/index.php/rbie/article/view/2851>.

ZHANG, L.; FU, K.; LIU, X. Artificial intelligence in education: Ethical issues and its regulations. In: **Proceedings of the 5th International Conference on Big Data and Education**. New York, NY, USA: Association for Computing Machinery, 2022. (ICBDE '22), p. 1–6. ISBN 9781450395793. Disponível em: <https://doi.org/10.1145/3524383.3524406>.