A Comparative Study on ChatGPT and Checklist as Support Tools for Unit Testing Education

Zihan Fang
zihan.fang@vanderbilt.edu
Vanderbilt University
USA

Jiliang Li
ericlij@stanford.edu
Stanford University
USA

Anda Liang
anda.liang@vanderbilt.edu
Vanderbilt University
USA

Gina R. Bai
rui.bai@vanderbilt.edu
Vanderbilt University
USA

Yu Huang
yu.huang@vanderbilt.edu
Vanderbilt University
USA

Abstract

Testing is widely practiced in software engineering, and many tools have been developed to support students in learning testing. Prior research suggests that a lightweight testing checklist improves learning outcomes but doesn't address students' challenges in writing test code that matches their intentions or design. Meanwhile, generative AI tools (e.g., ChatGPT) bring new promise as another form of software assistance tool. In this study, we examined the impact of various support tools (checklist, ChatGPT, or both) on unit testing among 42 students. Our results indicated that using these tools individually or in combination produced a comparable effect on student performance in unit testing. Students preferred using the checklist but acknowledged ChatGPT's effectiveness in accelerating task completion and addressing programming language challenges. While ChatGPT demonstrated potential benefits for testing education, it did not overcome the implementation challenges identified in the previous study. Moreover, reliance on ChatGPT may hinder students' deeper engagement with new concepts, which is crucial for comprehensive learning, as they often interact superficially with AI-generated responses without employing the critical thinking necessary to evaluate the information provided. Therefore, we proposed recommendations for both students and instructors on adapting to learning and teaching in the AI era and offer insights into the evolving role of AI in education.

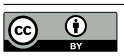
CCS Concepts

• **Applied computing** → **Education**; • **Software and its engineering** → *Software verification and validation.*

Keywords

ChatGPT, Checklist, Unit Testing, Testing Education

ACM Reference Format:
Zihan Fang, Jiliang Li, Anda Liang, Gina R. Bai, and Yu Huang. 2025. A Comparative Study on ChatGPT and Checklist as Support Tools for Unit Testing Education. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3696630.3727244



FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3727244

1 Introduction

Testing is a common practice in software engineering [1, 2]. Consequently, software testing education has garnered significant attention [3–5], leading to an increasing trend of incorporating testing education into standard computer science curricula [6–8]. When students learn about software testing, their performance can be improved with the help of various external testing support tools, such as coverage [9–11] and inquiry-based conceptual information [12]. A recent study by Bai et al. proposes a lightweight support tool for testing education, a testing checklist, that points at exactly what each test case and suite should aim to achieve [6]. The checklist has significant positive impacts on student performance in software testing [13], demonstrates comparable effectiveness to other advanced tools (e.g., EclEmma), and can be easily used by students without a steep learning curve [6]. Yet, although existing testing tools are often effective in suggesting what and where to test, researchers have observed that students may struggle to implement correct test code that accurately reflects their testing intentions or designs [14].

On this front, ChatGPT, powered by generative AI, has gained widespread recognition for its ability to generate code from natural language descriptions, offering new potential as a software assistance tool [15–17]. It provides numerous benefits, including improved coding efficiency, automated code generation, and enhanced code comprehension assistance [18, 19]. However, there is an ongoing debate about ChatGPT's reliability (e.g., generating correct, compilable code) and its potential to encourage superficial engagement [16, 20, 21]. Furthermore, in an educational context, ChatGPT often decreases the desire of students to explore traditional learning support resources by offering more easily accessible assistance [22].

Given these concerns, it is still unclear whether generative AI tools (e.g., ChatGPT) can enhance students' performance in software testing compared to traditional support tools (e.g., checklists) and resolve challenges that traditional tools cannot, such as implementation barriers. Therefore, a comprehensive evaluation of the impact of these support tools on educational outcomes in testing contexts is crucial for providing valuable insights to educators and

students and for advancing AI tools in software engineering education. To this end, we conducted a comparative study evaluating the impact of using a checklist [6], ChatGPT, or a combination of both on 42 students within the context of unit testing education. Our goal is to determine whether the checklist and ChatGPT produce comparable effects on students' unit testing performance. Furthermore, we aim to examine whether ChatGPT provides further benefits when used alongside the checklist, such as reducing barriers to implementing test cases that the checklist alone may not address. We claim the following contributions:

- A comprehensive investigation on the impact of using the checklist, ChatGPT, or both on students in unit testing.
- Present empirical evidence on effective tools for software testing education.
- Insights into developing better teaching practices and strategies for software testing.

Our results showed no statistically significant differences in task performance among students using the checklist, ChatGPT, or both tools combined. However, ChatGPT's effectiveness heavily depended on the quality of the prompts students provided. While students found ChatGPT useful for completing tasks more quickly and overcoming programming language challenges – evidenced by screen recordings of their interactions – it did not address the implementation challenges identified in prior research [14]. Furthermore, ChatGPT may even hinder deeper student engagement during implementation testing. Based on these findings, we proposed recommendations for both students and instructors on adapting to learning and teaching in the AI era and provided insights into the evolution of education-focused AI. All study materials are available online[1].

2 Background and Related Work

2.1 Software Testing Education

The critical role of testing in software development has heightened attention on testing education, essential for enhancing overall computer science training as students who write their test cases tend to produce higher-quality code [3–5, 23, 24]. As research on testing education progresses, studies have demonstrated the effectiveness of various pedagogical techniques in enhancing students' learning outcomes in software testing [25, 26]. Examples include peer testing [25] and cross-course activities [26]. Among these techniques, one prominent approach is providing support tools that provide information such as missing test cases and coverage [10, 11]. However, such tools tend to deliver ground-truth information devoid of insights into the fundamental testing concepts that render a test suite inadequate, and thus discourage critical thinking and encourage reliance on automated feedback [12, 27]. Moreover, studies have shown that senior-level CS students often struggle to effectively use coverage-based testing tools, despite these being recognized as state-of-the-art practices [6, 7]. In response, Cordova et al. [12] propose inquiry-based conceptual feedback given students' testing results. Recently, Bai et al. focus on the testing process and further devise a checklist pointing out what each test case and suite should achieve based on fundamental testing concepts and JUnit tutorials [6]. The checklist, easily transferable across university classrooms, has shown comparable learning outcomes to coverage tools in experiments [6]. However, previous research has also shown that the checklist does not assist students with implementation issues, and students often fail to implement correct test code that aligns with their test intentions or designs [14]. Therefore, there remains a need to find solutions to address implementation barriers in testing.

[1] https://anonymous.4open.science/r/TestingEdu-C5E5/

2.2 Test Quality Measurements

Test code quality is often assessed through metrics such as statement coverage and branch coverage, both measuring completeness [5, 8, 28]. However, such metrics have significant drawbacks as previous research indicates code coverage does not correlate with the effectiveness of test suites [29]. Additionally, it is possible to manipulate these metrics by writing inadequate tests that simply execute code without implementing effective assertions [8]. As a result, effectiveness metrics like mutation coverage have been adopted [1, 28, 30, 31], as well as measuring a test suite's ability to detect intentionally seeded defects [6, 14]. Besides these metrics, requirement coverage is used to assess how well the software meets its specified requirements [6, 14]. Maintainability is frequently evaluated by detecting test smells [14, 32], as their presence can cause inaccurate test outcomes during program evolution and hinder the maintainability of unit tests [33]. A widely adopted tool for assessing maintainability is SonarQube [34–37], an open-source static code analysis platform that identifies code smells to detect maintainability issues in software systems. In this study, we employed all metrics above to assess the quality of students' written tests thoroughly.

2.3 ChatGPT in CS Education

As a leading example of state-of-the-art generative AI, ChatGPT's impact on CS education has been extensively explored [38–40]. For example, Quereshi [41] demonstrates that students using ChatGPT achieve higher scores in a programming challenge, and Lyu et al. [42] further show that semester-long exposure to ChatGPT can improve student scores. Kosar et al. [40] conclude that it is safe for novice programmers to use ChatGPT, as its use did not have a significant impact on their learning and engagement. However, concerns regarding the use of ChatGPT in CS education also exist. Jalil et al. [16] found that ChatGPT could answer testing-related questions with only 53.0% partial accuracy. Another concern is about over-reliance on AI tools in software development education and the importance of educating students on ethical and critical AI use [43]. Additionally, Xue et al. [22] report that the availability of ChatGPT could drastically reduce students' use of other available resources. Prior research emphasizes the importance of guiding using generative AI and integrating it into courses, rather than allowing unsupervised use by students, which can negatively affect their learning [17]. The ambivalent role of ChatGPT in CS education motivates this study to explore its potential as an alternative or complementary testing support tool alongside the checklist [6], which serves as a benchmark for assessing ChatGPT's impact on testing education.

3 Study Design

We investigate how students use different tools (i.e., the checklist, ChatGPT, or a combination of both) to design and implement unit tests, and evaluate the effects of these tools on their performance, focusing on the following research questions:

- **RQ1:** How do students interact with different tools for unit testing tasks?
- **RQ2:** How does the use of different tools impact students' objective performance in unit testing?
- **RQ3:** What are students' subjective experiences with different tools for unit testing?

3.1 Student Participants

We recruited 42 students from a senior-level software engineering course at *institutions redacted for review*, all majoring in computer science. On average, student participants had 2.8 years of programming experience and 1.3 years of experience with Java. All participants were new to unit testing, with the majority (71.1%) self-identifying as novices or less skilled, while the remaining 28.9% classified themselves as advanced beginners. The average experience of the students in ChatGPT was 0.8 years. Initially, students were randomly assigned into three equally sized groups of 14 participants each. Students in Group 1 (G1) were required to use only the checklist, Group 2 (G2) used only ChatGPT, and Group 3 (G3) were required to use both the checklist and ChatGPT. However, three students did not follow the pre-assigned group requirements as evidenced in their screen recordings (e.g., students in G3 used only one tool instead of both), necessitating a redistribution of student participants across groups to maintain accurate analysis, as further discussed in Section 3.5. This experiment was conducted as part of the student's homework, contributing up to 10% of their final grade based on performance. To further motivate students to do their best, we awarded an additional one point toward their final course grade (out of 100) to those who ranked in the top 10% of their group.

Table 1: Demographic information of students in the final group assignment, along with their self-reported experience with Java, unit testing, and ChatGPT.

Groups	Education		Experience (yrs)		
	Undergrad	*Grad*	*Java*	*JUnit*	*ChatGPT*
Checklist (G1)	13	3	1.1	0.8	0.9
ChatGPT (G2)	13	2	1.3	1.0	0.8
Checklist & ChatGPT (G3)	9	2	1.7	0.9	0.8
Overall	35	7	1.3	0.9	0.8

3.2 Checklist for Unit Testing

We employed a lightweight testing support tool, the checklist, designed on the premise that while students understand software testing principles, they require guidance on specifics such as exception testing syntax [14]. The checklist addresses common difficulties students encounter during software testing, such as tests having syntax errors or lacking assertions [5,14], tests containing smells [5], premature termination of testing [44], test suites insufficiently covering boundary values and other program requirements [5, 44], and misinterpreting failing tests or modifying tests to remove the appearance of a failure [26]. Consequently, the checklist is designed to mitigate these challenges and is divided into two main sections: one for individual test cases (**Test Case Checklist**) and the other for the entire test suite (**Test Suite Checklist**). Each section contains two lists: one specifying essential actions that the test case or suite must perform, and another suggesting best practices to enhance testing effectiveness. The checklist is not designed to teach testing methodologies; rather, it serves as a prompt for students to incorporate various testing strategies. Following is the checklist that student participants referred to during testing:

Test Case Checklist

Each test case should:

- ☐ Be executable (i.e., it has an **@Test** annotation and can be run via "Run as JUnit Test")
- ☐ Have at least one assert statement or assert an exception is thrown. Example assert statements include: **assertTrue**, **assertFalse**, and **assertEquals** (click for tutorials). For asserting an exception is thrown, use **assertThrows** in JUnit 5 (click for tutorials).
- ☐ Evaluate/test only one method

Each test case could:

- ☐ Be descriptively named and commented
- ☐ If there is redundant setup code in multiple test cases, extract it into a common method (e.g., using **@BeforeEach**)
- ☐ If there are too many assert statements in a single test case (e.g., more than 5), you might split it up so each test evaluates one behavior.

Test Suite Checklist

The test suite should:

- ☐ Have at least one test for each requirement
- ☐ Appropriately use the setup and teardown code (e.g., **@BeforeEach**, which runs before each **@Test**)
- ☐ Contains a fault-revealing test for each bug in the code (i.e., a test that fails)
- ☐ For each requirement, contain test cases for:
  - ☐ Valid inputs
  - ☐ Boundary cases
  - ☐ Invalid inputs
  - ☐ Expected exceptions

To improve the test suite, you could:

- ☐ Measure code coverage using an appropriate tool, such as EclEmma (installation, tutorial). Inspect uncovered code and write tests as appropriate.

3.3 Tasks

The study consists of three primary tasks: two testing tasks adapted from a prior testing-related study [14], and a post-task survey. The first testing task requires students to design test suites *in comments*

based on specified requirements, while the second task involves implementing unit tests *in Java*. Both tasks offer sample cases in either natural language or JUnit format, emphasizing strict adherence to the provided specifications.

3.3.1 Test Design task - Mars Rover API. The goal of this task is to test an API that tracks a rover's movements on a simulated 100 x 100 grid and logs obstacles. Students will use a provided behavior specification to create basic test cases, each including a test name, scenario description, input specifications, and expected outcomes—all in comments. This non-coding task is designed for students with limited Java or JUnit experience and should be completed in about 20 minutes.

3.3.2 Test Implementation task - Bowling Score Keeper. The goal of this task is to evaluate an application designed to compute the score of a single bowling game. Students are provided with 1) a comprehensive description detailing the expected behaviors, and 2) a fully developed program containing three classes (totaling 86 lines of code) and three total seeded faults. However, students are not required to correct any defects or modify the source code. Instead, they are tasked with developing JUnit tests to assess the application's performance based on the provided specifications. The suggested completion time of this task is approximately 60 minutes.

3.3.3 Post-task Survey. The post-task survey includes multiple-choice, Likert-scale, and open-ended questions. It is structured to collect information on students' demographics (1[2]-7), programming proficiency (8-11), familiarity with JUnit and ChatGPT (12-15), and their perspectives on the effectiveness of various tools for unit testing (16-32). It also includes a question to verify tool usage (18), which serves as a cross-check to ensure that students in each group adhered to the tool usage requirements. In total, the survey comprises 32 questions, with certain questions tailored to specific student groups as necessary. The survey is administered through Google Forms and can be completed in approximately 5 minutes.

3.4 Protocol

The study was carried out as one of the assignments for a senior-level software engineering course. However, students had the option to include or exclude their assignment data from the study without any effect on their final grade. The IRB review is exempt because our local IRB has classified this study as a course quality improvement initiative. Before the assignment, instructors delivered two lectures on JUnit 5, as students were required to use it for all unit testing tasks. As mentioned in Section 3.1, students were randomly assigned to three groups. Thus, assignment instructions were placed in three separate GitHub repositories to distribute the materials, each corresponding to one of the groups. The teaching staff distributed assignment links via email, ensuring that the group assignment process and group members remained anonymous for each student participant. Each email contained a GitHub link to the assignment tasks and reiterated the tool requirements for the group. Specifically, G1 has been instructed in both email and task descriptions on GitHub to exclusively use the checklist, with a clear emphasis on avoiding any generative AI tools. G2, on the other hand, had the checklist removed from their task descriptions on GitHub and was explicitly directed to use only ChatGPT. G3 was instructed to use both ChatGPT and the checklist, with these instructions provided through both email and the GitHub repository. Apart from these differences, all other aspects of the experimental setup were identical across groups. To minimize unnecessary burdens and ensure equitable access, we did not mandate a specific ChatGPT version for students. However, we documented the version each student used during data annotation. Students were given a two-week window to complete the assignment at their convenience. While suggested completion time was provided for each task (as mentioned in Section 3.3), students had the flexibility to finish within or beyond the suggested time. Additionally, students were required to record their computer screens throughout the assignment and complete a post-survey upon completion. The experimental setup and protocol are summarized in Fig. 1, and all materials used in the study are available online.

3.5 Group Reassignment

As mentioned in Section 3.1, initially, we randomly and evenly assigned students to three groups and then confirmed that their prior experience with Java, JUnit, and ChatGPT was not significantly different between the groups. However, since three students did not strictly follow the instructions (e.g., students in G3 did not refer to the checklist or ChatGPT), we adjusted the group distribution slightly based on their observed tool usage in the screen recordings. After the reassignment, G1 consisted of 16 students, G2 had 15 students, and G3 included 11 students. To further validate the accuracy of the group reassignment—particularly due to the absence of valid screening recordings from five student participants, as noted in Section 3.6.1—we analyzed students' responses to a post-survey question about the tools they used during the testing tasks. This analysis confirmed that the reassignment accurately reflected their actual behavior. Additionally, to ensure comparability among the three groups after reassignment, we analyzed their midterm scores, Java experience, ChatGPT experience, and unit testing experience, finding no statistically significant differences between groups. Table 1 summarizes the demographic information of eligible student participants in the final group assignment.

3.6 Data Analysis

3.6.1 Screen recordings. We collected approximately 55.5 hours of screen recordings from 37 participants, as five of the 42 participants either failed to submit their recordings or provided invalid submissions due to technical issues. The recordings were distributed across three groups, with 15 participants in G1, 13 in G2, and 9 in G3. The duration of individual recordings varied, ranging from 28 to 142 minutes. Three authors manually annotated the screen recordings to thoroughly log each participant's actions during the testing tasks, including actions such as *refer to the checklist*, *use ChatGPT-4.0*, *review source code*, and *develop test cases*. To ensure accuracy, each annotated action was precisely timestamped and thoroughly described, adhering to best practices in qualitative analysis [45]. As noted in 3.4, the task instructions did not specify which version

[2]1 refers to the first question of the post-task survey provided in the supplementary material, with this numbering format consistently applied to all subsequent questions. For example, 1–32 indicates the complete set of 32 survey questions.

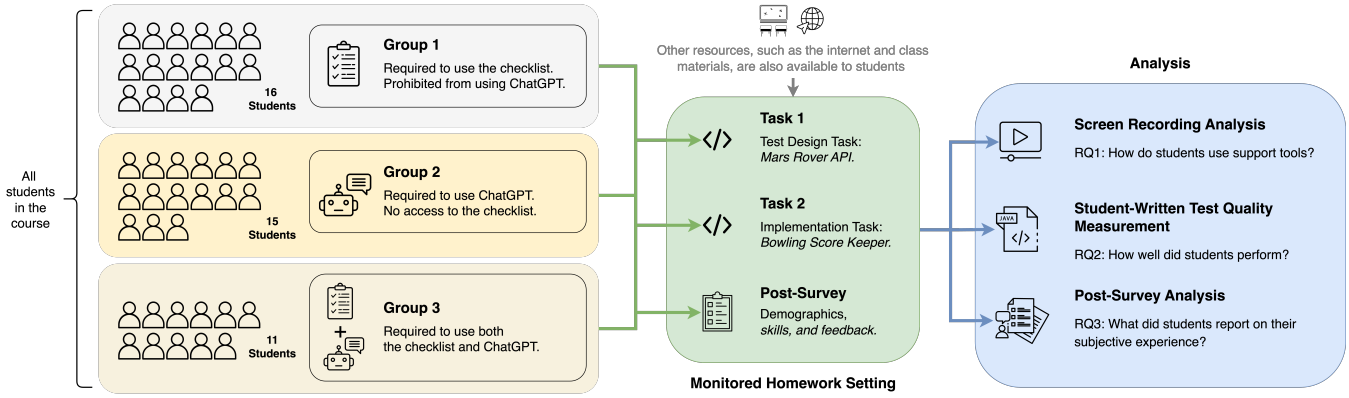


Figure 1: Group assignments, experiment protocols, and analysis procedures illustration

of ChatGPT students should use. However, the screen recordings confirmed that all valid student submissions used ChatGPT-4.0.

Initially, one researcher drafted a preliminary codebook based on three recordings from three groups. The manual annotation process involved a comprehensive consensus-building approach to ensure reliability [46]. Both agreement and disagreement cases were systematically recorded: instances of disagreement often stemmed from overlapping or ambiguous code definitions and were addressed through iterative discussions, revisiting the original screen recording, and iterative revisions involving all authors, culminating in a final codebook. To ensure the reliability of the final codebook, the first author conducted validation using recordings from three additional participants, confirming its stability and applicability. The codebook is publicly accessible online.

3.6.2 Unit Testing Tasks. Inspired by previous studies [6, 14], we evaluated students performance from four main aspects:

Completeness: Measured using *Requirements Coverage*, *Instruction Coverage* and *Branch Coverage*. We manually calculated the requirement coverage to ensure that all functional and non-functional specifications were covered, regardless of whether they were implemented correctly. Additionally, we employed EclEmma[3] to calculate instruction and branch coverage to assess how thoroughly the different logical paths of the application were tested.

Effectiveness: Measured using *Mutation Coverage* and *Identified Bug*. *Mutation Coverage* quantifies the percentage of simulated faults (mutants) detected by the test suite [47] and was calculated using PITest[4]. Moreover, as stated in Section 3.3.2, we introduced three seeded bugs into the source code [14] in the second testing task to simulate real-world debugging scenarios. We then manually verified the number of tests written by students that successfully exposed these seeded bugs [14].

Maintainability: Assessed through *Maintainability Issues* and *Test Name Quality*. *Maintainability Issues*, identified using SonarQube[5]—a popular open-source static analysis tool—are measured by the number of code smells detected based on violations of rules defined in SonarQube's default Quality Profile. Complementing this, *Test Name Quality* measures the clarity, descriptiveness, and meaningfulness of the test names, which were manually rated by one author on a scale from 1 (low quality) to 5 (high quality). These ratings were subsequently verified by another author to ensure their accuracy and validity.

Implementations Barrier: Measured using *Normalized Erroneous Implementations*, indicating the proportion of test cases that stray from the student's original intent by failing to meet either programming standards or logic. This metric is calculated by dividing the number of *Erroneous Test Cases* by the *Total Test Count* per student. The *Total Test Count* includes all test cases written by a student, regardless of correctness. An *Erroneous Test Cases* is defined by meeting any of the following criteria: (1) the test case fails to compile; (2) the test case does not match its specified description in the header comments; (3) the test case exhibits flawed logic, such as using incorrect bowling rules as the oracle or applying both `assertTrue` and `assertFalse` on the same Boolean condition; (4) the test throws errors due to inadequate exception handling.

3.6.3 Statistical Analysis. A total of forty-two valid testing script submissions were collected for both the test design and test implementation tasks. Two-way ANOVA was performed to analyze the metrics outlined in Section 3.6.2 for evaluating testing results, aiming to assess the impact of different support tools on students' testing performance [48]. Additionally, post-task survey data from forty-two valid responses across three groups were analyzed. The Mann-Whitney U Test was used to examine differences between two groups with non-normally distributed data [49], while the Wilcoxon Signed-Rank Test was applied for within-group comparisons across different tool usages to account for non-normality [50].

4 Results

We discuss how students use the checklist and/or ChatGPT for unit testing, along with their test performance and subjective experiences with these tools.

[3]https://www.eclemma.org/
[4]https://pitest.org/
[5]https://www.sonarsource.com/products/sonarqube/

4.1 RQ1: How do students interact with different tools for unit testing?

Conclusion Analysis of students' interactions with the tools revealed that ChatGPT could provide comprehensive support for unit testing tasks, enabling students to rely less on traditional resources (e.g., task descriptions and tutorials) and complete test cases more efficiently than when using only the checklist. However, the benefits of using ChatGPT for unit testing largely depend on students' ability to engage with the tool effectively, particularly in crafting high-quality prompts.

Table 2: Average time (in minutes) spent on each action by students in three groups. * indicates the significant difference confirmed by ANOVA.

Actions	Average Time in Minutes		
	Checklist (G1)	*ChatGPT (G2)*	*Checklist & ChatGPT (G3)*
Test Design task - Mars Rover API (Task 1)			
1) ChatGPT	-	7.4	5.6
2) Checklist	1.1	-	1.3
3) Google	0.9	0	1.5
4) Other Resources	3.5	5.4	1.2
5) Total Working Time	16.8	12.1	18.8
6) Time Per Test	3.1	2.7	3.1
Test Implementation task - Bowling Score Keeper (Task 2)			
1) ChatGPT	-	16.5	10.7
2) Checklist	1.2	-	1.4
3) Google	2.3	6.3	0.9
4) Other Resources *	13.8	5.8	2.3
5) Total Working Time	38.7	35.6	30.1
6) Time Per Test	4.3	4.2	3.3

In general, ChatGPT can offer students more comprehensive support during testing tasks than the checklist. Students with access to ChatGPT were less likely to rely on traditional resources, such as task descriptions and tutorials, for both designing and implementing testing tasks. Specifically, 93.8% of G1 students have referred to traditional resources, in contrast to 73.3% and 72.7% for G2 and G3, respectively. Additionally, G1 students (50.0%) were more likely to use Google when compared to G2 and G3 students (20.0% and 27.3% respectively) in both tasks. While the observed difference ($p = 0.31$, $\eta^2 = 0.19$) was not statistically significant, it suggested a potential trend where support from ChatGPT may reduce students' reliance on Google and other external resources for testing tasks.

This finding is further illustrated by the average time spent on each action during testing, as summarized in Table 2. For the test design task, the time spent on actions across all groups is comparable. However, for the test implementation task, students in G2 and G3, who were allowed to use ChatGPT, spent significantly less time using other external resources compared to students in G1 ($p = 0.0003$, $\eta^2 = 0.41$), with average times of 13.8 minutes for G1, 5.8 minutes for G2, and 2.3 minutes for G3. Moreover, we found that students in groups G2 and G3 who used ChatGPT often accelerated their test implementation by copying and pasting the generated test cases. Notably, 77.3% of these students either failed to run the tests generated by ChatGPT or accepted the results without reviewing or verifying the underlying logic. In contrast, G1 students, who were restricted to using only the checklist, generally spent more time writing and revising unit test cases compared to G2 and G3, with average durations of 38.7, 35.6, and 30.1 minutes, though the differences were not statistically significant ($p = 0.44$, $\eta^2 = 0.04$). Additionally, no significant differences were found among the groups in the time spent on other actions, such as using ChatGPT or using the checklist. However, with the support of ChatGPT, students generally completed the task more quickly, as shown in Table 2.

Although the reduced time with ChatGPT suggested increased efficiency, the reliability of this method remained inconsistent. Our analysis of ChatGPT prompts revealed that students who generated effective test cases with ChatGPT were inclined to provide complete task descriptions and all relevant source codes. This approach aligns more closely with the traditional software testing method of designing test cases first and then implementing them. However, students who provided incomplete information in the prompt faced challenges in obtaining useful responses from ChatGPT. For example, one student's prompt to ChatGPT was:

> *"Given this program: [Frame.java][6] how can I test it?"*

In response, ChatGPT primarily generated test cases that invoked the Frame constructor, leading the student to struggle until additional source files were incorporated into the prompts. Moreover, another representative, yet ineffective prompt was:

> *"Did you cover all possible cases?"*

Students often used this prompt toward the end of their interaction with ChatGPT, which typically responded with numerous redundant test cases that did not increase code coverage. Therefore, the extent of the benefits students can derive from ChatGPT was heavily dependent on their ability to interact effectively with the tool, suggesting the importance of not only technical skills but also the ability to clearly and comprehensively articulate problems and requirements in prompts.

4.2 RQ2: How does the use of different tools impact students' objective performance in unit testing?

Conclusion The support tools (checklist, ChatGPT, or both) have comparable effects on student performance in testing tasks. While ChatGPT can help clarify complex requirements and encourage more strategic test designs, leading to fewer unnecessary test cases, its ability to address implementation challenges is limited and may foster superficial engagement.

In general, our evaluation metrics showed that the three types of supports provided–the checklist, ChatGPT, or both–demonstrated similar efficiency in improving student performance in testing tasks with no statistically significant differences. Table 3 presents the average unit testing performance of student participants in each group across different tasks

[6] A Java class in the source code for the test implementation task

For *Test Design task - Mars Rover API*, we assessed students' designed tests in terms of completeness (e.g., Requirement Coverage), maintainability (e.g., Test Name Quality), and Total Test Count. Requirement coverage improved in the groups that used ChatGPT—G2 with 80.0% coverage and G3 with 82.7%—compared to the group that exclusively used the checklist (G1), which achieved 74.3% coverage. However, this difference was not statistically significant ($p = 0.26$, $\eta^2 = 0.58$). Interestingly, despite having lower coverage, G1 produced a higher total number of test cases, averaging 11.2 tests, compared to 8.5 and 8.7 tests in G2 and G3, respectively. While this difference was also not significant ($p = 0.39$, $\eta^2 = 0.49$), the finding that students using ChatGPT achieved higher requirement coverage with fewer tests suggested that ChatGPT may help clarify complex requirements or encourage more strategic decisions about what needs to be tested. This, in turn, could lead to more efficient test designs that cover more requirements with fewer test cases.

For the *Test Implementation task - Bowling Score Keeper*, we assessed the student-written test code in terms of completeness (e.g., Requirements, Instruction, and Branch Coverage), effectiveness (e.g., Mutation Coverage and Identified Bugs), maintainability (e.g., Maintainability Issues and Test Name Quality), and implementation barriers (e.g., Normalized Errors), as summarized in Table 3. In terms of test effectiveness and maintainability, students in G2 and G3 generally outperformed those in G1. However, G1 showed a potential to surpass G2 and G3 in completeness metrics. Nonetheless, these differences across the three groups were not statistically significant.

Additionally, since the checklist is unable to effectively address implementation barriers [14], we hypothesize that ChatGPT, with its strength in code generation, may help reduce the difficulties students face in writing code implementations that accurately reflect their intentions. However, students with access to ChatGPT in groups G2 and G3 generated nearly twice as many test cases that deviated from their original intentions, whether in terms of programming standards or logic. This trend is evident in the normalized error rates, with G1 at 0.2 and both G2 and G3 at 0.4, as presented in Table 3. Moreover, this finding is consistent with our screen-recording analysis in Section 4.1, which revealed that 77.3% of students using ChatGPT either did not execute a GPT-generated test case or accepted the results without verifying the logic of the generated tests. Notably, four students from each of G2 and G3 submitted completely non-compilable codes, compared to only two in G1. These findings suggested that ChatGPT's influence on overcoming implementation barriers may be limited and may even promote superficial task engagement.

4.3 RQ3: What are students' subjective experiences with different tools for unit testing?

Conclusion When both the checklist and ChatGPT are available, students prefer the checklist. Compared to the checklist, students found that ChatGPT offers less guidance on determining the adequacy of a test suite. However, ChatGPT is effective in helping them complete test cases more quickly and in resolving programming language-related challenges.

Table 3: Assessing student code in Test Design task and Test Implementation Task. Different colors are used to distinguish various aspects evaluated: Completeness, Effectiveness, Maintainability, and Implementation Barriers.

Metrics	Groups (AVG)		
	Checklist (G1)	*ChatGPT (G2)*	*Checklist & ChatGPT (G3)*
Test Design task - Mars Rover API (Task 1)			
1) Requirements Cov (%)	74.3	88.0	82.7
3) Test Name Quality	4.5	4.5	4.2
2) Total Test Count (#)	11.2	8.5	8.7
Test Implementation task - Bowling Score Keeper (Task 2)			
1) Requirements Cov (%)	76.4	86.7	91.6
2) Instruction Cov (%)	68.0	60.0	51.8
3) Branch Cov (%)	61.2	54.5	45.4
4) Mutation Cov (%)	53.3	64.3	61.0
5) Identified Bugs (#)	1.4	1.2	2.0
6) Maintainability Issues (#)	10.1	13.9	11.5
7) Test Name Quality	3.9	4.1	4.3
8) Erroneous Test Cases (#)	4.3	9.4	8.0
9) Total Test Count (#)	18.8	22.7	19.3
10) Normalized Errorenous	0.2	0.4	0.4

While completing the testing tasks, we evaluated students' perceptions of the effectiveness of different testing support tools by examining their usage patterns, encountered challenges, and support needs. We also assessed how helpful they perceived each tool to be and their likelihood of recommending it for testing purposes.

Results from the post-task survey indicated that students perceived both ChatGPT and the checklist as providing similar guidance on what to test next during testing tasks[7]. However, compared to the checklist, ChatGPT may not offer equal insights into when a test suite is complete and when students should stop writing additional test cases. Specifically, G1 students relied more on the checklist than G2 students using ChatGPT to decide when to stop testing (33.3% vs. 8.3%). This was further confirmed in G3, where students had access to both tools but still favored the checklist over ChatGPT (46.7% vs. 6.7%) when deciding to stop testing.

Moreover, challenges in testing faced by students[8] across all groups included *time constraints*[9], *IDE Setup*, *producing more correct test cases*, *task background understanding*, *programming syntax*, and *tool utilization*. Notably, *producing more correct test cases* was the most prevalent challenge. Additionally, *time constraints* were notably significant for G1, where 26.7% of students reported such issues. The same issue was less common in groups using ChatGPT (G2 and G3), suggesting that ChatGPT may alleviate time-related challenges and enhance overall efficiency in testing. However, while it offered advantages, students still highlighted specific challenges in using ChatGPT, such as:

[7] refer to survey question 19, 23, 26, 30.
[8] refer to survey question 17.
[9] Time constraints: we provided suggested completion times for each task in the instructions, but students struggled to complete them within the recommended time frame.

> *"It is hard to use ChatGPT because writing test relates to several java files."*
>
> *"ChatGPT tended to hallucinate on the math, leading to unexpected failures."*

Furthermore, the survey responses revealed that most students expressed a desire for additional support[10], specifically in providing more detailed *task descriptions*, help with *IDE setup*, and guidance on *programming* during the testing task. Interestingly, we observed that ChatGPT significantly aided students in resolving programming issues. In G1, 53.3% of students reported needing help with programming challenges, while only 16.7% in G2 and 13.3% in G3 indicated requiring assistance with programming for testing. More broadly, integrating both ChatGPT and the checklist may provide more effective support for students with potential testing issues—26.7% of students in G3 reported that they did not need any additional help during the tasks, a result not observed in the other groups. Additionally, based on students' survey responses, we observed that using ChatGPT may boost their confidence in the quality and effectiveness of the test suites required by the checklist specifications[11]. Students in G1 estimated that 77.0% of their test cases followed the checklist specifications, whereas G3 reported an average compliance of 88.0%. The Mann-Whitney U Test confirmed a significant difference between the two groups ($p = 0.008$, $r = -0.44$).

Lastly, we examined students' perceptions of the helpfulness of the checklist and/or ChatGPT[12] and whether they would recommend these tools to future students for unit testing[13]. Notably, students in G3 favored the checklist over ChatGPT when both tools were available, with a higher likelihood of recommending it to future students, awarding it an average score of 4.4 out of 5 compared to 3.9 out of 5 for ChatGPT ($p = 0.04$, $r = -0.53$). However, interestingly, such a difference is not observed between G1 vs. G2 – when only one tool is available, G1's ratings on the checklist were similar to G2's ratings on ChatGPT (4 vs. 4).

5 Threats to Validity

5.1 Construct

The study evaluated students' performance using different tools. Thus, students' prior familiarity with ChatGPT or the checklist may have influenced their effective interaction, making it challenging to isolate the tools' effects from their previous experience. However, as outlined in Section 3.2, the checklist consists of straightforward, easy-to-understand text and requires no prior experience or steep learning curve to use effectively, which helps mitigate the impact of familiarity on the results.

Furthermore, our experiment focused on assessing students' immediate performance with or without generative AI tools through direct observations, rather than evaluating their long-term knowledge retention. The use of generative AI tools (e.g., ChatGPT) might encourage students to rely on quick answers rather than developing a deeper understanding of concepts, leading to surface-level learning rather than mastery of the related knowledge. As a result, our approach and findings may not fully capture students' lasting understanding or their ability to apply knowledge over time when using various tools. Recognizing that knowledge retention is vital for effective learning, we suggest future research explore the impact of generative AI on students' retention of knowledge.

5.2 Internal

As mentioned in Section 3.6.1, while we initially recruited 42 participants, only 37 screen recordings were deemed valid after excluding the invalid data. The small sample size may have contributed to the lack of statistically significant results in Section 4.1, limiting the study's ability to detect meaningful differences or patterns. Furthermore, several studies have highlighted that the quality of prompts affects the effectiveness of generative AI tools [51, 52]. The variability in students' ability to craft effective prompts may therefore impact the tool's overall utility, which is further discussed in Section 6.

Furthermore, the timing of the study–conducted as homework immediately before a holiday–could have affected the level of effort exerted by the students, potentially affecting their performance and participation in the tasks. Such factors should be considered when interpreting the findings and assessing their applicability to other contexts.

5.3 External

In this study, all participants were recruited from the same class and were novices in both Java and unit testing, which may limit the generalizability of the findings. The homogeneity of the sample could reduce the diversity needed to apply the results to broader populations and might introduce biases tied to specific educational practices within that class. Moreover, the shared experiences among students may limit the variability required for robust statistical analysis, while peer dynamics could influence individual responses, potentially skewing the results further.

In addition, all students completed the experiment remotely, which posed challenges in controlling the environment and ensuring compliance with group-specific requirements (e.g., exclusive checklist use by G1). While students were instructed to record their screens, they could still use additional devices or seek external help, potentially affecting the results. However, since the experiment was part of their course assignments, students were informed that they must follow the honor code, adhere to the instructions, and avoid collusion. We also manually reviewed the screen recordings to ensure that all students completed the tasks in a reasonable amount of time and in an appropriate manner (as outlined in Section 3.4, where the suggested completion time is provided in the introduction). Additionally, to mitigate this limitation, the post-task survey included a question to confirm which tools students used during the testing tasks.

6 Discussion

6.1 Guidelines for Students to Improve Interaction with AI for Learning and Testing

As discussed in Section 4.1, our analysis of screen recordings revealed that the quality of ChatGPT prompts drafted by students

[10] refer to survey question 16.
[11] refer to survey question 20, 27.
[12] refer to survey questions 21, 24, 28, 31.
[13] refer to survey question 22, 25, 29, 32.

largely influenced the effectiveness of the generated test suites. Poorly constructed prompts, such as those providing only partial information, made it more challenging for students to apply the generated content to their tasks. This finding aligns with previous research in other fields on different tasks [51, 52]. Effectively communicating with AI tools is becoming a crucial skill both now and in the future [53]. Specifically, to generate meaningful and effective unit tests with AI tools, students should provide detailed and specific prompts that clearly outline the task and its context. Vague prompts often lead to incomplete or irrelevant test cases, as the AI lacks sufficient information to understand the program's logic. Including relevant portions of the source code in the prompt helps the AI better grasp the functionality, resulting in more accurate and targeted test cases. For example, instead of asking, *"How can I test this program?"*, a more effective prompt would be, *"How can I write unit tests for the* `calculateTotal()` *method in* `Frame.java` *to handle edge cases like negative inputs or null values?"*. This level of detail directs the AI to concentrate on specific aspects of the code.

Additionally, students should avoid broad and ambiguous questions such as *"Did you cover all cases?"*, which often results in redundant and unhelpful test cases. A more effective strategy is to specify the exact scenarios or edge cases they intend to test. For instance, asking, *"What test cases should I write for the* `validateInput()` *method to handle invalid user inputs?"* prompts the AI to generate more relevant and comprehensive suggestions. This targeted approach not only produces better results but also highlights the importance of crafting high-quality prompts—an ability that depends on a strong understanding of the code and testing principles. Without foundational knowledge, students may struggle to formulate effective prompts, limiting the value of AI-generated tests. Therefore, while AI tools can assist in the testing process, mastering software testing concepts still remains crucial for developing effective and thorough prompts.

Moreover, trusting AI-generated content without thorough evaluation can introduce logical or syntactical errors into students' testing tasks. Rather than accepting ChatGPT's responses uncritically, students should carefully assess the information it provides. Specifically, students should view AI-generated output as a preliminary reference and verify its accuracy by consulting their knowledge, textbooks, or other reliable sources to ensure correctness and deepen their understanding. For example, they can use checklists to verify that GPT-generated test cases align with established criteria, critically evaluating and validating the provided responses. However, as discussed in Section 4.1, our findings indicated that with the support of ChatGPT, students tended to rely less on traditional resources such as task descriptions, tutorials, and checklists. This shift may hinder their ability to fully comprehend tasks, potentially weakening their critical thinking and problem-solving skills. Therefore, in the era of generative AI, students need to develop the skills to use AI tools effectively and responsibly rather than relying on them uncritically. For instance, students can engage with AI tools by asking it to explain concepts to test their understanding or to present alternative perspectives. Additionally, they can use AI tools to refine their problem-solving strategies, ensuring active engagement with the process rather than depending on AI for complete solutions.

In addition, in Section 4.3, students highlighted ChatGPT's effectiveness in enabling them to complete testing tasks more efficiently. While AI tools can support students in managing their study time by providing quick clarifications and minimizing time spent on minor obstacles, it is still important for students to balance this efficiency with intentional practice to avoid compromising deep and thorough learning for the sake of speed. To achieve this, establishing clear learning objectives before using AI can help students stay focused on mastering concepts rather than merely completing assignments.

6.2 Guidelines for Educators on Integrating AI into Education

As discussed above, the negative outcomes associated with generative AI tools (e.g., ChatGPT) often stem from poorly constructed prompts. Effectively using ChatGPT, therefore, requires proficiency in prompt engineering—the skill of crafting precise and relevant questions. Students who lack this ability may find ChatGPT less beneficial. To address this challenge, educators could offer training on effective interaction with AI tools, focusing on defining clear context, specifying necessary information, and outlining the desired scope of responses for specific software engineering tasks. This need also underscores the importance of incorporating AI communication skills into future curricula, much like the current emphasis on coding and technical writing.

However, a critical concern arises: if students become adept at prompt engineering and rely on generative AI tools to produce accurate and comprehensive answers, they may prioritize using the tool over acquiring foundational knowledge. To mitigate this risk, educators should thoughtfully incorporate generative AI tools through strategies that promote deeper learning. For instance, assignments could require students to use generative AI tools for self-assessment by comparing their solutions with AI-generated responses to identify gaps in understanding. Additionally, educators may encourage reflective learning by having students critique and refine AI-generated content, explaining why certain solutions are effective or flawed. By embedding AI tools into activities that foster critical thinking and self-reflection, educators can shift students' focus from merely using the tool to leveraging AI for meaningful and enriching learning experiences.

Moreover, educators should carefully consider when to encourage the use of generative AI and when its use may be less beneficial. As shown in our findings in Section 4.2, although ChatGPT led to higher requirement coverage, students faced more implementation challenges compared to those who used only the checklist. This outcome may stem from the checklist's effectiveness in guiding students through tasks with clear, well-defined requirements by providing a structured, step-by-step framework that fosters critical thinking and independent coding, rather than reliance on pre-generated solutions. In contrast, while ChatGPT can supply students with code, the generated content may not always align with specific testing tasks. Nevertheless, ChatGPT proves valuable for tasks requiring exploration and creativity by offering diverse perspectives and enabling rapid content generation. Therefore, educators should establish clear criteria to determine when to use generative AI tools, traditional support methods (e.g., checklists),

or a combination of both for specific tasks, ensuring that each tool is strategically integrated to maximize its effectiveness and improve student learning efficiency and outcomes.

6.3 Evolution of Generative AI in Education

In Section 4.3, we found that students generally preferred using checklists over ChatGPT when both were available for unit testing tasks. This preference may be attributed to the perceived reliability of checklists, which are specifically designed for the testing process and are particularly valuable in contexts where the accuracy of the test suite is critical, such as in professional settings. Thus, the structured and well-validated format of checklists offers consistent and dependable guidance. In contrast, although ChatGPT provides broad insights, its responses can vary in relevance and accuracy, often being more general and not always directly addressing specific issues. A previous semester-long study demonstrated that an education-focused large language model (LLM) can significantly improve student learning outcomes [42]; however, a general-purpose AI tool like ChatGPT may not yield similar results. This highlights the need to develop generative AI tools with specialized educational features tailored to software engineering or computer science, which could offer greater benefits in the future. For instance, in software testing, a promising improvement could involve hybrid approaches that combine AI-generated insights with checklists, merging the concise structure of checklists with the expansive knowledge base of AI.

Moreover, there remains a need for AI-driven personalized tutoring systems to provide customized feedback and guidance to students. Further research in this area could facilitate the development of education-focused tools that cater to diverse learning styles and subject-specific needs, ultimately enhancing student performance.

In addition, since neither the checklist nor ChatGPT fully resolves the implementation challenges students face in testing, future research could investigate additional solutions to better support students in overcoming these barriers. One promising direction involves advancing education-focused AI tools to not only support task completion but also enhance the overall learning experience. To maintain the integrity of the learning process, AI solutions designed for education could be developed to provide more effective learning aids, strategies, and resources, potentially contributing to reduced disparities in educational access. Future research could explore how educational AI tools can be enhanced to promote deeper learning, despite challenges associated with prompt design. For example, these tools could be designed to pose thought-provoking questions that stimulate critical thinking and encourage students to engage more deeply with the material, positioning AI as an active partner in the learning process.

7 Conclusion

Software testing education has garnered considerable attention and is steadily being integrated into computer science curricula. Numerous tools have been developed to improve student performance, learning experiences, and outcomes. In parallel, ChatGPT offers new potential as an additional software assistance tool. In this study, we explored the effects of different support tools—a traditional checklist, ChatGPT, and a combination of both—on students' performance in unit testing tasks within software engineering. The results indicated no statistically significant differences in performance outcomes among students using these tools. However, the availability of ChatGPT could provide more comprehensive support to students during testing, reducing their dependence on other external resources and improving their time efficiency in implementing test cases. While the integration of generative AI, such as ChatGPT, into software testing education holds promising benefits, it does not address the implementation challenges previously identified in prior research [6]. One possible explanation is that students often engage with AI-generated responses superficially, without applying critical thinking to evaluate or reflect on the generated content. Additionally, the quality and accuracy of the generated test cases are largely influenced by students' ability to formulate effective prompts. We then proposed recommendations for students on how to effectively incorporate generative AI into their learning process and for educators on how to support students in adapting to the evolving AI landscape. Furthermore, we proposed potential avenues for the development of AI tools tailored specifically to educational contexts.

8 Acknowledgements

We thank the support of NSF for the study: this research was supported in part by NSF grants CCF-2211429 and IUSE-2141923.

References

[1] Giovanni Grano, Fabio Palomba, and Harald C Gall. Lightweight assessment of test-case effectiveness using source-code-quality indicators. *IEEE Transactions on Software Engineering*, 47(4):758–774, 2019.

[2] Leandro Sales Pinto, Saurabh Sinha, and Alessandro Orso. Understanding myths and realities of test-suite evolution. In *Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering*, pages 1–11, 2012.

[3] Vahid Garousi, Austen Rainer, Per Lauvås Jr, and Andrea Arcuri. Software-testing education: A systematic literature mapping. *Journal of Systems and Software*, 165:110570, 2020.

[4] Lilian Passos Scatalon, Jeffrey C Carver, Rogério Eduardo Garcia, and Ellen Francine Barbosa. Software testing in introductory programming courses: A systematic mapping study. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 421–427, 2019.

[5] Maurício Aniche, Felienne Hermans, and Arie Van Deursen. Pragmatic software testing education. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 414–420, 2019.

[6] Gina R Bai, Kai Presler-Marshall, Thomas W Price, and Kathryn T Stolee. Check it off: Exploring the impact of a checklist intervention on the quality of student-authored unit tests. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, pages 276–282, 2022.

[7] Jeffrey C Carver and Nicholas A Kraft. Evaluating the testing ability of senior-level computer science students. In *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*, pages 169–178. IEEE, 2011.

[8] Sarah Heckman, Jessica Young Schmidt, and Jason King. Integrating testing throughout the cs curriculum. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 441–444. IEEE, 2020.

[9] Jim Collofello and Kalpana Vehathiri. An environment for training computer science students on software testing. In *Proceedings Frontiers in Education 35th Annual Conference*, pages T3E–6. IEEE, 2005.

[10] Jaime Spacco, David Hovemeyer, William Pugh, Fawzi Emad, Jeffrey K Hollingsworth, and Nelson Padua-Perez. Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. *ACM Sigcse Bulletin*, 38(3):13–17, 2006.

[11] Stephen H Edwards. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 26–30, 2004.

[12] Lucas Cordova, Jeffrey Carver, Noah Gershmel, and Gursimran Walia. A comparison of inquiry-based conceptual feedback vs. traditional detailed feedback mechanisms in software testing education: an empirical investigation. In *Proceedings of the 52nd ACM Technical symposium on computer science education*, pages 87–93, 2021.

[13] Gina R Bai, Zuoxuan Jiang, Thomas W Price, and Kathryn T Stolee. Evaluating the effectiveness of a testing checklist intervention in cs2: A quasi-experimental replication study. In *Proceedings of the 20th ACM Conference on International Computing Education Research V.1 (ICER '24 Vol. 1)*, New York, NY, USA, August 13–15 2024. ACM.

[14] Gina R Bai, Justin Smith, and Kathryn T Stolee. How students unit test: Perceptions, practices, and pitfalls. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 248–254, 2021.

[15] Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li. Gptutor: a chatgpt-powered programming tool for code explanation. In *International Conference on Artificial Intelligence in Education*, pages 321–327. Springer, 2023.

[16] Sajed Jalil, Suzzana Rafi, Thomas D LaToza, Kevin Moran, and Wing Lam. Chatgpt and software testing education: Promises & perils. In *2023 IEEE international conference on software testing, verification and validation workshops (ICSTW)*, pages 4130–4137. IEEE, 2023.

[17] Marian Daun and Jennifer Brings. How chatgpt will change software engineering education. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, pages 110–116, 2023.

[18] Paramarshi Banerjee, Anurag Srivastava, Donald Adjeroh, Y Ramana Reddy, and Nima Karimian. Understanding chatgpt: impact analysis and path forward for teaching computer science and engineering. *Authorea Preprints*, 2023.

[19] Stephen MacNeil, Andrew Tran, Dan Mogil, Seth Bernstein, Erin Ross, and Ziheng Huang. Generating diverse code explanations using the gpt-3 large language model. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 2*, pages 37–39, 2022.

[20] Long Bai, Xiangfei Liu, and Jiacan Su. Chatgpt: The cognitive effects on learning and memory. *Brain-X*, 1(3):e30, 2023.

[21] A Alshahrani. The impact of chatgpt on blended learning: Current trends and future research directions. *International Journal of Data and Network Science*, 7(4):2029–2040, 2023.

[22] Yuankai Xue, Hanlin Chen, Gina R Bai, Robert Tairas, and Yu Huang. Does chatgpt help with introductory programming? an experiment of students using chatgpt in cs1. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, pages 331–341, 2024.

[23] Michael K Bradshaw. Ante up: A framework to strengthen student-based testing of assignments. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 488–493, 2015.

[24] Otávio Augusto Lazzarini Lemos, Fábio Fagundes Silveira, Fabiano Cutigi Ferrari, and Alessandro Garcia. The impact of software testing education on code reliability: An empirical assessment. *Journal of Systems and Software*, 137:497–511, 2018.

[25] Alessio Gaspar, Sarah Langevin, Naomi Boyer, and Ralph Tindell. A preliminary review of undergraduate programming students' perspectives on writing tests, working with others, & using peer testing. In *Proceedings of the 14th annual ACM SIGITE conference on Information technology education*, pages 109–114, 2013.

[26] Upsorn Praphamontripong, Mark Floryan, and Ryan Ritzo. A preliminary report on hands-on and cross-course activities in a college software testing course. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 445–451. IEEE, 2020.

[27] Kevin Buffardi and Stephen H Edwards. Reconsidering automated feedback: A test-driven approach. In *Proceedings of the 46th ACM Technical symposium on computer science education*, pages 416–420, 2015.

[28] Stephen H Edwards and Zalia Shams. Comparing test quality measures for assessing student-written tests. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 354–363, 2014.

[29] Laura Inozemtseva and Reid Holmes. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th international conference on software engineering*, pages 435–445, 2014.

[30] Jeffrey Voas. How assertions can increase test effectiveness. *IEEE Software*, 14(2):118–119, 1997.

[31] Zalia Shams and Stephen H Edwards. Toward practical mutation analysis for evaluating the quality of student-written software tests. In *Proceedings of the ninth annual international ACM conference on International computing education research*, pages 53–58, 2013.

[32] Kevin Buffardi and Juan Aguirre-Ayala. Unit test smells and accuracy of software engineering student test suites. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 234–240, 2021.

[33] Davide Spadini, Martin Schvarcbacher, Ana-Maria Oprescu, Magiel Bruntink, and Alberto Bacchelli. Investigating severity thresholds for test smells. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 311–321, 2020.

[34] Hen Kian Jun and Muhammad Ehsan Rana. Evaluating the impact of design patterns on software maintainability: An empirical evaluation. In *2021 Third International Sustainability and Resilience Conference: Climate Change*, pages 539–548. IEEE, 2021.

[35] Rolf-Helge Pfeiffer and Mircea Lungu. Technical debt and maintainability: How do tools measure it? *arXiv preprint arXiv:2202.13464*, 2022.

[36] Arthur-Jozsef Molnar and Simona Motogna. A study of maintainability in evolving open-source software. In *Evaluation of Novel Approaches to Software Engineering: 15th International Conference, ENASE 2020, Prague, Czech Republic, May 5–6, 2020, Revised Selected Papers 15*, pages 261–282. Springer, 2021.

[37] Ana Díaz Muñoz, Moisés Rodríguez Monje, and Mario Gerardo Piattini Velthuis. Towards a set of metrics for hybrid (quantum/classical) systems maintainability. *JUCS: Journal of Universal Computer Science*, 30(1), 2024.

[38] Ishika Joshi, Ritvik Budhiraja, Harshal Dev, Jahnvi Kadia, Mohammad Osama Ataullah, Sayan Mitra, Harshal D Akolekar, and Dhruv Kumar. Chatgpt in the classroom: An analysis of its strengths and weaknesses for solving undergraduate computer science questions. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 625–631, 2024.

[39] Carlos Alexandre Gouvea da Silva, Felipe Negrelle Ramos, Rafael Veiga de Moraes, and Edson Leonardo dos Santos. Chatgpt: Challenges and benefits in software programming for higher education. *Sustainability*, 16(3):1245, 2024.

[40] Tomaž Kosar, Dragana Ostojić, Yu David Liu, and Marjan Mernik. Computer science education in chatgpt era: Experiences from an experiment in a programming course for novice programmers. *Mathematics*, 12(5):629, 2024.

[41] Basit Qureshi. Chatgpt in computer science curriculum assessment: An analysis of its successes and shortcomings. In *Proceedings of the 2023 9th International Conference on e-Society, e-Learning and e-Technologies*, pages 7–13, 2023.

[42] Wenhan Lyu, Yimeng Wang, Tingting Rachel Chung, Yifan Sun, and Yixuan Zhang. Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study. *arXiv preprint arXiv:2404.13414*, 2024.

[43] Olga Petrovska, Lee Clift, Faron Moller, and Rebecca Pearsall. Incorporating generative ai into software development education. In *Proceedings of the 8th Conference on Computing Education Practice*, pages 37–40, 2024.

[44] Stephen H Edwards and Zalia Shams. Do student programmers all tend to write the same software tests? In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 171–176, 2014.

[45] Kay E Ramey, Dionne N Champion, Elizabeth B Dyer, Danielle T Keifert, Christina Krist, Peter Meyerhoff, Krystal Villanosa, and Jaakko Hilppö. Qualitative analysis of video data: Standards and heuristics. Singapore: International Society of the Learning Sciences, 2016.

[46] Clara E Hill, Barbara J Thompson, and Elizabeth Nutt Williams. A guide to conducting consensual qualitative research. *The counseling psychologist*, 25(4):517–572, 1997.

[47] James H Andrews, Lionel C Briand, Yvan Labiche, and Akbar Siami Namin. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering*, 32(8):608–624, 2006.

[48] Lars St, Svante Wold, et al. Analysis of variance (anova). *Chemometrics and intelligent laboratory systems*, 6(4):259–272, 1989.

[49] Joachim Krauth. The interpretation of significance tests for independent and dependent samples. *Journal of neuroscience methods*, 9(4):269–281, 1983.

[50] Bernard Rosner, Robert J Glynn, and Mei-Ling T Lee. The wilcoxon signed rank test for paired comparisons of clustered data. *Biometrics*, 62(1):185–192, 2006.

[51] Yunlong Wang, Shuyuan Shen, and Brian Y Lim. Reprompt: Automatic prompt editing to refine ai-generative art towards precise expressions. In *Proceedings of the 2023 CHI conference on human factors in computing systems*, pages 1–29, 2023.

[52] Cheng Chen, Sangwook Lee, Eunchae Jang, and S Shyam Sundar. Is your prompt detailed enough? exploring the effects of prompt coaching on users' perceptions, engagement, and trust in text-to-image generative ai tools. In *Proceedings of the Second International Symposium on Trustworthy Autonomous Systems*, pages 1–12, 2024.

[53] Tira Nur Fitria. Artificial intelligence (ai) in education: Using ai tools for teaching and learning process. In *Prosiding Seminar Nasional & Call for Paper STIE AAS*, volume 4, pages 134–147, 2021.