Received 7 February 2025, accepted 21 February 2025, date of publication 26 February 2025, date of current version 20 March 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3545882

RESEARCH ARTICLE

Exploring the Integration of Generative AI Tools in Software Testing Education: A Case Study on ChatGPT and Copilot for Preparatory Testing Artifacts in Postgraduate Learning

SUSMITA HALDAR[1], (Member, IEEE), MARY PIERCE[2], AND LUIZ FERNANDO CAPRETZ[3], (Senior Member, IEEE)

[1]School of Information Technology, Fanshawe College, London, ON N5Y 5R6, Canada
[2]Faculty of Business, Information Technology and Part-Time Studies, Fanshawe College, London, ON N5Y 5R6, Canada
[3]Department of Electrical and Computer Engineering, Western University, London, ON N6A 3K7, Canada

Corresponding author: Susmita Haldar (shaldar@fanshawec.ca)

ABSTRACT Software testing education is important for building qualified testing professionals. To ensure that software testing graduates are ready for real-world challenges, it is necessary to integrate modern tools and technologies into the curriculum. With the emergence of Large Language Models (LLMs), their potential use in software engineering has become a focus, but their application in software testing education remains largely unexplored. This study, conducted in the Capstone Project course of a postgraduate software testing program, was carried out over two semesters with two distinct groups of students. A custom-built Travel Application limited to a web platform was used in the first semester. In the second semester, a new set of students worked with an open-source application, offering a larger-scale, multi-platform experience across web, desktop, and mobile platforms. Students initially created preparatory testing artifacts manually as a group deliverable. Following this, they were assigned an individual assignment to generate the same artifacts using LLM tools such as ChatGPT 3.5 in the first semester and Microsoft Copilot in the second. This process directly compared manually created artifacts and those generated using LLMs, leveraging AI for faster outputs. After completion, they responded to a set of assigned questions. The students' responses were assessed using an integrated methodology, including quantitative and qualitative assessments, sentiment analysis to understand emotions, and a thematic approach to extract deeper insights. The findings revealed that while LLMs can assist and augment manual testing efforts, they cannot entirely replace the need for manual testing. By incorporating innovative technology into the curriculum, this study highlights how Generative AI can support active learning, connect theoretical concepts with practical applications, and align educational practices with industry needs.

INDEX TERMS Capstone project, ChatGPT, generative AI, software testing education, Microsoft Copilot, sentiment analysis.

I. INTRODUCTION

This study investigates how large language models (LLMs), such as ChatGPT and Microsoft Copilot, can be incorporated into software testing education. Software testing is a crucial step in delivering software products, but it often involves labor-intensive tasks, particularly when creating test cases, mapping requirements using a requirements traceability matrix (RTM), and writing test scripts. Alshahwan et al. [1] identified several software testing challenges from an industry perspective. For regression testing, determining the number of test cases or prioritizing test cases to execute is

The associate editor coordinating the review of this manuscript and approving it for publication was Nkaepe Olaniyi.

critical, especially when addressing non-functional requirements like performance or security alongside functional verification. In the case of unit testing, even minor changes in system behavior can lead to test failures, making it difficult to maintain test cases over time. For end-to-end testing, understanding the cascading impact of a single change across interconnected components is a key challenge, highlighting the need for a comprehensive dependency analysis. Additionally, they emphasized that advancements in artificial intelligence hold significant potential for addressing these challenges.

Software Engineering Body of Knowledge (SWEBOK) [2] highlights the importance of software testing for both functional and non-functional requirements. It also notes that creating test cases can be time-consuming and often requires automation to be effective. To address these challenges, two different semesters were examined in this study, each with its own set of postgraduate students. In the first semester, students worked with a custom travel application that offered limited capabilities. In the second semester, an open-source application was introduced, providing more complex features and allowing validation of the teaching techniques. Previous research has shown that open-source projects can support practical, hands-on software engineering education [3], and more specifically software testing education [4].

As industry practices evolve, educational approaches must also adapt to ensure that students gain the skills needed in modern testing contexts. Recent curriculum guidelines like CS2023 [5] underline the growing role of artificial intelligence (AI) in computer science education. In addition, technical reports such as the Future of the Workforce (2022) [6] and Technology Predictions (2024) [7] further emphasize the rapid rise of Generative AI, reinforcing its significance for modern testing education.

Large language models are being adopted in the educational context [8]. ChatGPT [9] and Microsoft Copilot [10], [11] are examples of LLM-driven tools that can understand, generate, and summarize the text in a human-like manner. This study investigates how these tools can create preparatory testing artifacts, such as test cases and scripts, without giving students direct access to source code.

For communicating with LLM models to generate the required outputs, a prompt needs to be constructed which is written in natural language text describing the task that an AI should perform [12]. Prompt engineering is a methodology of structuring and optimizing an instruction that can be interpreted and understood by a generative artificial intelligence [13]. Various prompt engineering techniques have been introduced in the literature. For this study, the author found that the iterative tuning of the prompts worked the best, and the students were instructed that providing more details and being specific would help with achieving the best outcome.

The Capstone Project, offered as a mandatory course during the second four-month term (Level 2) of the post-graduate certificate program in Software Testing, requires students to develop use cases, a requirements traceability matrix, test cases, and test scripts for a specified software system. After completing these assignments, they compare their manually generated documents with those created by the selected LLM tool. This comparison helps them evaluate whether AI-based testing methods should be adopted and identify any necessary precautionary measures, contributing to their learning process.

By focusing on test design, functional versus non-functional testing, problem-solving, and effective use of digital tools, this learning activity addresses key course outcomes. It also meets vocational objectives that include planning and executing testing for various technologies and approaches, such as web applications, mobile platforms, and agile frameworks.

The following questions guide the study:

- RQ1 Can a single teaching approach be effective for all testing types, or should different contexts have specialized methods?
- RQ2 What role should AI-based testing play in the curriculum, and how should it be implemented?
- RQ3 Does incorporating LLMs into testing education help students develop skills that align with industry needs?

By exploring how LLM tools like ChatGPT and Microsoft Copilot can enrich the teaching of software testing, this study aims to propose effective strategies for bridging traditional methods with emerging, AI-driven practices.

The contribution of this work can be summarized as:

- Expanded upon a preliminary study [14] by increasing the scope from two questions evaluated with a count-based approach to 11 questions analyzed using diverse and extensive evaluation strategies.
- Introduced multi-platform testing in the second term by incorporating an open-source application, enabling comparative analysis of ChatGPT and Copilot for enhanced depth and broader applicability.
- Developed an innovative assignment framework integrating Generative AI tools with multiple evaluation approaches, including count-based, thematic, and sentiment analysis to critically assess AI tools' strengths and limitations in generating and evaluating preparatory testing artifacts.
- Provided students with realistic, hands-on learning experience through open-source, multi-platform applications, fostering critical thinking and practical skills development.
- Contributed to the Scholarship of Teaching and Learning by creating a replicable model for integrating AI tools into software testing education, demonstrating their impact on student learning outcomes.

The rest of this paper follows this organization. Section II shows the background and literature review for this study. Next, section III describes the methodology and evaluation strategies followed by section V, where the result of this work is presented. Afterward, this study follows the analysis and discussion, threads to validity, and the conclusion of the presented work.

II. BACKGROUND AND LITERATURE SURVEY

Before actual test execution, preparation for effective test execution and monitoring strategies needs to be defined ahead of time. Test cases are often generated from requirements or use cases of the application being validated. Madan et al. [15] explained the importance of the Requirements Traceability Matrix (RTM), which links or tracks the mapping between two different artifacts, such as requirements and test cases, in the context of web application testing. However, they highlighted that generating RTMs is time-consuming, requiring clear and detailed documentation of requirements and corresponding test cases. Building on this, earlier research by Uusitalo et al. [16] emphasized the challenges of implementing complete test traceability to requirements, recommending improved communication links between testers and requirement owners to overcome deficiencies in document-based links.

To address these challenges, this study explores the potential of Generative AI tools to automate such processes, providing students with an opportunity to apply these techniques manually while critically evaluating their results. Wang et al. [17] supported this idea, arguing that applying natural language processing (NLP) to use case specifications allows testers to systematically produce acceptance test cases that are both comprehensive and aligned with original requirements. Similarly, Naimi et al. [18] presented an approach for automatic test case generation from use case diagrams using large language models (LLMs) and prompt engineering. Their work focused on generating use cases that include main flows and alternative flows to help testers understand the system's behavior comprehensively. Building on these approaches, this study extends by exploring how LLMs assist in generating use cases and test cases.

In the software testing paradigm, the application of LLM tools has been widely studied in various activities [19], [20] including unit test generation [21], [22], regression test case generation [23], test script generation [22], and impact analysis of requirement changes [24]. For instance, Wang et al. [25] conducted a comprehensive review of LLM utilization in software testing and identified test case preparation and program repair as two of the most representative tasks in this domain. These studies highlight the role of LLM tools in automating labor-intensive preparatory testing tasks and improving efficiency and accuracy.

Despite the increasing application of LLMs in testing preparation activities, their utilization in software testing education remains relatively unexplored. A notable gap exists between the skills required for real-world assignments and what is currently taught in testing courses. For instance, Aludhilu and Sutinen [26] discussed the disparity between industry expectations and the software testing training provided in Namibian startup companies. Similarly, Garousi et al. [27] highlighted knowledge gaps in software testing according to industry needs, emphasizing the importance of incorporating measures or metrics beyond basic code coverage to assess the quality of software testing education. Siddique et al. [28] conducted a study that highlighted the software industry needs and trends related to the freelancing industry and uncovered suggestions for training in this dynamic field.

As LLMs gain popularity in practical applications, their integration into software engineering education is becoming increasingly inevitable [29]. The possibility of integrating LLM tools have been investigated in multiple educational domains, including entrepreneurship education [30], healthcare [31], learning of human-computer information techniques [32], and software engineering education, including requirements engineering [33] and coding assistants [34]. Ramos et al. [35] demonstrated how effective lesson planning and assessment design can be achieved using Microsoft Copilot Implementation. However, they also highlighted limitations and challenges in incorporating Copilot into lesson planning, such as the need to address AI biases ensure ethical considerations like data privacy and establish accountability for AI mistakes. They further recommend that educators receive training and support to optimize the use of these tools.

Karmela et al. [36] compared four Generative AI tools which are ChatGPT-3.5, ChatGPT-4, Google Bard, and Microsoft Copilot in the context of enhancing educational engagement. They surveyed 246 students and 105 educators and used ten evaluation criteria, such as accuracy, prompt speed, plug-ins, and data confidentiality. Although GPT-4 excelled in accuracy, empathy, plug-in support, and handling diverse input/output formats, all tools displayed strengths and weaknesses. Bard and Copilot offered real-time web information and linking but had limitations in plug-in ecosystems and language handling. The authors recommend using multiple AI tools, verifying outputs with traditional methods, and crafting precise prompts to maximize the benefits of Generative AI while ensuring rigor and accuracy.

These studies underscore the potential of tools like ChatGPT in transforming educational practices. This study builds on these findings, investigating how ChatGPT and similar LLM tools Microsoft Copilot can enhance software testing education and bridge the gap between industry requirements and academic training.

Evaluating qualitative and quantitative data in software testing education involves various methodologies, including count-based approaches for analyzing frequencies, sentiment analysis to assess emotional tone, and thematic analysis for identifying underlying patterns and themes in student responses. These approaches are essential for understanding the effectiveness of tools, pedagogies, and student outcomes in software testing education.

Thematic analysis identifies patterns and themes within textual data, providing insights into user experiences, perceptions, and learning outcomes. Rahman et al. [37] combined thematic analysis with natural language processing (NLP) to analyze competencies in software testing education. Similarly, Bilow and DeWaters [38] used thematic analysis to explore the connection between socio-technical engineering courses and students' sense of belonging and engineering identity, highlighting its utility in understanding broader educational themes.

Recent studies, such as Shoufan [39], utilized thematic analysis to explore students' perceptions of ChatGPT, categorizing their responses into recurring themes. This approach offers a systematic method for uncovering underlying patterns in textual feedback, making it particularly relevant for analyzing responses in software testing education.

Sentiment analysis, a popular method for classifying textual responses into Positive, Neutral, or Negative sentiments, has been extensively applied in education. Grimalt-Álvaro and Usart [40] conducted a systematic literature review on using sentiment analysis for formative assessment, emphasizing its ability to capture students' emotional and cognitive responses to educational tools. Dake and Gyimah [41] extended this work by applying sentiment analysis to evaluate qualitative student feedback, demonstrating its relevance in assessing perceptions of educational methods and technologies. Although sentiment analysis has been applied in broader educational contexts to evaluate perceptions of tools and methods, its specific application in software testing education remains unexplored. This study aims to employ sentiment analysis to assess student perceptions of Generative AI tools like ChatGPT and Copilot in artifact generation tasks.

Keyword extraction techniques are essential for analyzing structured and semi-structured data, particularly in educational and software documentation contexts. Liang et al. [42] developed Python-based software for automated keyword extraction, showcasing its effectiveness in identifying relevant terms in English texts. Similarly, Onan et al. [43] proposed an ensemble of keyword extraction methods for text classification, emphasizing its applicability in identifying critical terms and phrases within large datasets.

Beyond keyword extraction, text mining has been increasingly used to analyze open-ended questions and feedback. Buenaño-Fernandez et al. [44] utilized Latent Dirichlet Allocation (LDA) for topic modeling to analyze teacher self-assessments, illustrating the effectiveness of advanced text analysis techniques in educational contexts. Their approach demonstrates how automated tools can extract meaningful patterns from qualitative data, making it particularly relevant for evaluating open-ended responses in software testing education. Studies such as Grönberg et al. [45] demonstrated the use of text mining for analyzing student feedback, further validating the application of keyword extraction in educational research. In the context of software testing education, in this study, keyword extraction facilitated the identification of terms like 'use cases', 'RTM', and 'test cases', which are central to evaluating the completeness and accuracy of testing artifacts.

Enoiu [46] emphasized the challenges and best practices in teaching software testing to industrial practitioners using distance learning. This study highlighted the importance of integrating innovative evaluation techniques to assess the learning outcomes of software testing courses. The current study builds on these findings by using thematic analysis, sentiment analysis, and keyword extraction for count-based analysis to evaluate student responses to 11 analytical questions, providing a comprehensive understanding of the effectiveness of Generative AI tools in software testing education.

Although previous studies have explored these techniques separately, few have integrated them to evaluate Generative AI tools in software testing education. This study fills this gap by employing a mixed-methods approach to analyze responses from two student cohorts, comparing the effectiveness of ChatGPT-3.5 and Microsoft Copilot across multiple platforms. The combined use of these techniques allows for a more holistic understanding of the role of AI in artifact generation and software testing workflows.

III. METHODOLOGY

This study analyzed student responses from the Capstone Project course, part of the post-graduate certificate program in Software and Information Systems Testing at Fanshawe College, Canada. This second-semester course prepares students for real-world software testing challenges, followed by a co-op term. Foundational courses in Test Methodologies, Applied Project Management, and Systems Design and Analysis are prerequisites for the Capstone Project course. Students enrolled in the program must have either a two- to three-year college diploma, a business or information technology degree, or a combination of relevant work experience and post-secondary education.

This Capstone Project course emphasizes both theoretical and practical learning. Theoretical concepts are taught in parallel with project work, enabling students to apply what they learn to real-world scenarios. Working in teams of 4–5, students undertake a comprehensive software testing project, progressing through phases including test strategy development, test planning, artifact creation (use cases, requirements traceability matrices, test cases, and test scripts), test execution, and reporting. The initial 9 weeks focus on manually creating these artifacts, reinforcing foundational knowledge and logical reasoning in testing. Following this, as part of an in-class exercise during regular class hours, students individually generate the same artifacts using Generative AI tools within a 3-hour session.

A. TEACHING APPROACH

This course employed a structured teaching approach, combining lectures, practical demonstrations, independent assignments, and collaborative team activities. Weekly lectures introduced key software testing concepts and demonstrated tools applicable across various contexts. These were complemented by project management tasks and regular presentations to strengthen technical proficiency and professional skills.

The teaching approach was designed to align with the program's course and vocational learning outcomes, ensuring that students developed both technical proficiency and essential employability skills. Key goals included:

Course Learning Outcomes (CLOs)

- Demonstrating the creation of test cases and scenarios from application requirements.
- Differentiating between functional and non-functional testing types in theory and practice.

Essential Employability Skills (EES)

- Applying systematic and critical thinking to solve problems.
- Analyzing, organizing, and documenting information effectively.
- Taking responsibility for decisions and outcomes.

Vocational Learning Outcomes (VLOs)

- Developing test cases for various testing levels and types, including functional, usability, and regression testing.
- Designing testing protocols for specialized technologies such as web, mobile, and Agile framework applications.

In this 15-week course, this assignment was due in week 9. The following summarizes the course content leading up to this assignment.

- Weeks 1-3: Introduced software testing fundamentals, project planning, and setting up testing environments for the first cohort who used a web-only platform, and the second cohort who used an application that utilized desktop, web, and mobile platforms. Lectures covered configuring tools and understanding application contexts.
- Weeks 4–6: Teams developed and presented test strategies and detailed test plans. Feedback from these presentations helped refine their approaches.
- Weeks 7–9: Students created use cases, requirements traceability matrices (RTM), test cases, and test scripts. Milestone assessments evaluated their progress and artifact quality.
- Week 9: An assignment was introduced where students compared Generative AI tools, ChatGPT 3.5 in Term 1 and Microsoft Copilot in Term 2, to replicate artifacts created during the earlier weeks. This allowed for a direct comparison between manual and AI-assisted processes.

In addition to technical activities, students engaged in weekly project management tasks to simulate industry practices. These tasks included documenting project challenges, tracking team discussions through meeting notes, and providing weekly status reports to the professor, who acted as the business stakeholder. Regular presentations evaluated both technical artifacts such as installation and configuration of applications to test, test strategies, RTMs, and test cases, and soft skills like teamwork and communication. This approach ensured alignment with program outcomes and prepared students for real-world testing challenges.

B. ASSIGNMENT DESIGN AND STUDY FRAMEWORK

The study spanned two terms and included responses to 11 analytical questions designed to assess artifact coverage, accuracy, efficiency, and perceptions of AI tools. Table 1 lists these questions, and the associated rubrics are detailed in Table 2.

The first study involved 61 students submitting assignments, out of which 26 students who scored at least 90% were selected for further analysis. Students use a custom-built travel application as software under test this semester. Since ChatGPT-3.5 was free and widely used, it was selected as the large language model tool for this phase. Students responded to all 11 analytical questions during this term; however, the analysis focused on only two questions. Question 2 examined specific testing artifacts such as 'use cases', 'RTM', 'test cases', and 'test scripts' while Question 9 explored students' perceptions of ChatGPT-assisted testing, focusing on aspects like test case generation speed, coverage, accuracy, and ease of use. These findings, presented in prior work [14], highlighted ChatGPT's effectiveness in generating testing artifacts compared to manual efforts.

In this study, responses to all 11 questions from the first study were revisited and analyzed to gain a broader perspective. The dataset remained the same, as it included the original responses from the first cohort using the travel application. No changes were made to the assignment inputs or the exercise structure for this phase. By analyzing all 11 questions, this study aimed to provide a more comprehensive understanding of how students engaged with both manual and AI-assisted approaches using ChatGPT-3.5.

The second study, conducted in a subsequent term, introduced a new application, MoneyManagerEx [47], a mature open-source application available on web, desktop, and Android platforms. Unlike the travel application, MoneyManagerEx provided comprehensive documentation and a user manual [48] which included textual instructions, diagrams, and examples. This documentation enabled students to generate and evaluate testing artifacts in a multi-platform context effectively. Students could either extract relevant content manually from the user manual or upload sections of the document into Copilot, which assisted in generating artifacts through contextual prompts. This flexibility allowed students to leverage the quality and structure of the manual directly, providing sufficient input to facilitate precise and efficient artifact generation.

Microsoft Copilot was selected as the AI tool for this term due to its secure access features, including the option not to save prompts. A total of 35 students submitted assignments, and 21 students scoring 90% or higher were included in the analysis. This phase provided additional insights into AI

TABLE 1. Questions asked in the assignment.

Questions	Response Options
1. When compared to manually creating test cases as a group, did the selected LLM tool (ChatGPT/Copilot) provide you with additional information?	Free-response. Hints: a) Yes b) No or c) Yes and No
2. In which areas did the selected LLM tool work the best compared to you working as a group in generating the preparatory testing artifacts manually?	Free-response. Hints: Some possible areas include a) Test Cases b) Test Scripts c) Use Cases d) RTM, etc.
3. On a scale of 1 to 5 (1 being strongly disagreed and 5 being strongly agree), how would you rate the effectiveness of using ChatGPT/Copilot in generating responses compared to manual methods?	
4. What aspects were missing in your submitted test cases compared to the test cases generated from ChatGPT/Copilot? Please provide your findings.	
5. Among the test cases generated from ChatGPT/Copilot, which ones do you think won't apply or be feasible for you? How many test cases fall into this category, and why?	
6. Can this LLM tool be a beneficial addition to your overall testing process, and if so, how does it complement or enhance manual methods?	
7. What did you miss in your submitted test cases if you compared your test cases with the generated test cases from LLM?	
8. Do you still need this manual process of generating use cases, test plans, test scripts, and RTM? Please support your justification.	
9. Which aspect of ChatGPT/Copilot-assisted testing did you find most beneficial?	a) Test case generation speed b) Test case coverage c) Test case accuracy d) Ease of use e) Other (please specify).
10. How do you see the integration of ChatGPT/Copilot into your testing practices based on your experiences with both manual and AI-assisted methods? Please justify your answer.	
11. The selected LLM tool can entirely replace human effort in the process of creating test scripts, test cases, use cases, and RTM.	

TABLE 2. Rubrics used for evaluating students' submission.

Criteria	Excellent Response	Average Response	Poor Response
Identification of Missing Aspects	Clearly identifies multiple missing aspects and provides a detailed analysis	Identifies some missing aspects and provides analysis	Identifies few or no missing aspects
Depth of Analysis	Provides thorough analysis and reasoning.	Provides adequate analysis and reasoning.	Provides limited or no analysis.
Clarity of Explanation	The explanation is clear, concise, and coherent.	The explanation is clear and coherent.	The explanation is unclear or irrelevant.
Overall Understanding and Insight	Demonstrates deep understanding and insight.	Demonstrates good understanding.	Demonstrates limited understanding.
Identification of Inapplicable Test Cases	Clearly identifies specific test cases and provides a detailed explanation.	Identifies some specific test cases and provides an explanation.	Identifies few or no specific test cases and provides a limited explanation.
Explanation of Feasibility Issues	Provides detailed explanations for each identified test case.	Provides explanations for most identified test cases.	Provides limited or no explanations.
Justification of Findings	Justification is logical, comprehensive, and supported by evidence.	Justification is logical and supported.	Justification lacks logic or support.

tool performance by introducing platform diversity and more robust application documentation.

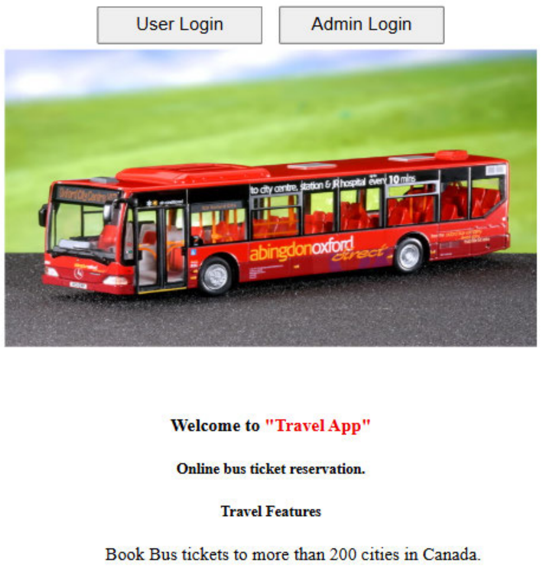
This dual-phase exercise enabled a direct comparison between manually created and AI-generated artifacts. Students were also given flexibility in tailoring prompts to include application-specific information, helping them navigate the non-deterministic nature of large language models (LLMs) for optimized results.

Figure 1 shows the user interface of the custom-built travel application used in the first study. Figure 2 illustrates the graphical user interface of the MoneyManagerEx application utilized in the second student cohort. The desktop version is shown in Figure 2a, the web application is illustrated in Figure 2b, and the mobile version is reflected in Figure 2c.

This study provides insights into the application of comparative analysis between different LLM tools and their effectiveness compared to manually generated artifacts, offering valuable lessons for their integration into software testing education.

C. EVALUATION METHODS

A mixed-methods approach was employed to analyze responses, integrating count-based analysis, sentiment analysis, and theme identification. Each method addressed specific aspects of the data. The count-based analysis quantified the presence of specific terms or features in structured and open-ended responses, helping to assess artifact completeness and identify patterns in artifact creation. Terms like use cases, test cases, test scripts, and RTM were mapped to evaluate the thoroughness of student submissions. For open-ended responses, sentiment analysis categorized student feedback into Positive, Neutral, or Negative sentiments using a standardized framework. This framework employed VADER (Valence Aware Dictionary and sEntiment Reasoner) [49], a rule-based sentiment analysis tool from the NLTK library, to compute polarity scores and classify sentiments based on predefined thresholds. This method provided insights into students' emotional responses and their perceptions of the tools. Theme identification focused on broader patterns and



Welcome to "Travel App"

Online bus ticket reservation.

Travel Features

Book Bus tickets to more than 200 cities in Canada.

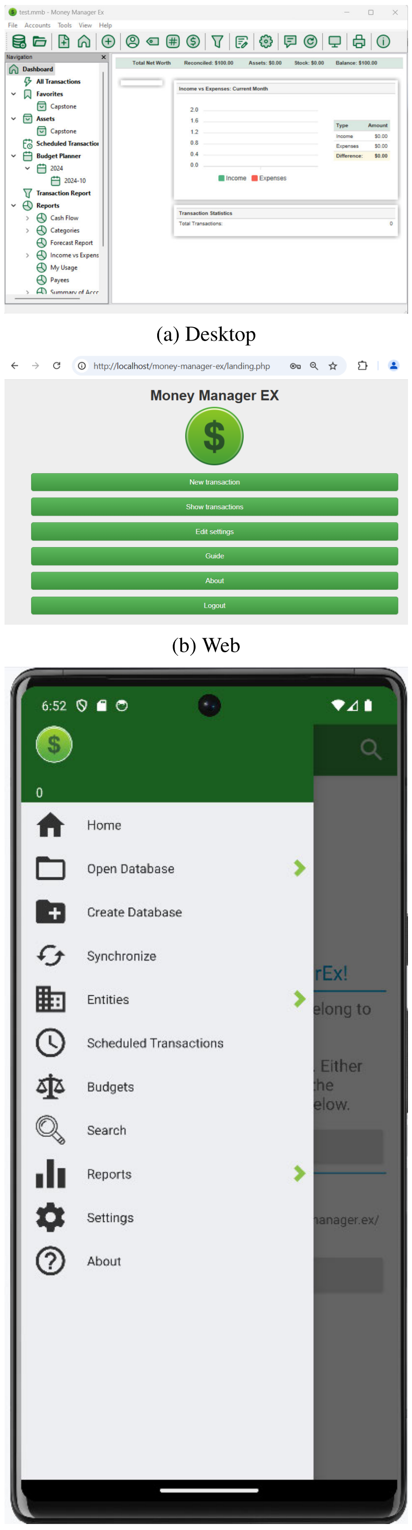
FIGURE 1. Custom travel application - supports web platform only.

justifications in responses, identifying themes such as 'AI as a Supplement', 'Human Oversight Required', and 'AI Limitations' to better understand the tools' impact on testing workflows.

This study compared two cohorts to explore the strengths and limitations of ChatGPT-3.5 and Microsoft Copilot in generating testing artifacts. ChatGPT was characterized by its simplicity and adaptability, making it easy to use across diverse scenarios. In contrast, Copilot leveraged multi-platform documentation and mature features for structured workflows. Both tools demonstrated unique strengths, and their combined analysis provided a comprehensive understanding of the role of Generative AI in software testing education.

The analytical questions were grouped based on the evaluation techniques most relevant to their objectives. The count-based analysis technique was applied to questions that required quantifying specific terms or features. For instance, Question 1 counted the number of students who thought the selected LLM tool either ChatGPT or Copilot added additional information compared to manual testing. Questions 2 and 3 focused on evaluating areas where ChatGPT or Copilot outperformed manual efforts. Question 2 was a free-response question with suggested options provided as hints, while Question 3 used a Likert scale rating from 1 (strongly disagree) to 5 (strongly agree) to assess effectiveness. Question 4 involved open-ended responses where keywords such as security and maintenance were manually extracted and mapped to increase the count, while Question 9 combined multiple-choice responses with 'other' options, where students' additional feedback was analyzed and mapped to specific keywords before automating the extraction process.

Sentiment analysis was applied to open-ended questions to explore students' perceptions of the tools. This analysis used



(a) Desktop

(b) Web

(c) Mobile

FIGURE 2. Money manager application - different platforms.

VADER from the NLTK library [50], which classified feedback into Positive, Neutral, or Negative sentiments based on polarity scores. For example, Question 6 examined whether Copilot enhanced the testing process and complemented manual methods. Question 8 investigated whether manual processes were still necessary for testing workflows, while Question 10 focused on the integration of LLM tools into testing workflows and its perceived impact.

Theme identification was employed for questions requiring a deeper understanding of students' justifications and reasoning. Question 5 analyzed key themes identifying patterns such as 'Irrelevant Features', 'Out of Scope', 'Features Not Available', 'lacking details', etc. whereas Question 7 analyzed key themes in responses to explore students' reasoning, identifying patterns such as 'non-functional tests', 'edge cases', etc. Question 11 combined techniques where the Yes/No responses were analyzed using count-based analysis and the accompanying justifications were mapped to thematic categories like 'AI Advantages' and 'Human Oversight' for additional qualitative insights.

The findings from these analyses were visualized through grouped bar charts and comparative plots. Count-based analyses highlighted quantitative differences, such as keyword frequencies and response distributions across tools. Sentiment and thematic analyses provided qualitative insights into students' perceptions and reasoning. For example, bar charts illustrated the frequency of specific keywords across ChatGPT and Copilot responses, while comparative plots highlighted differences in their strengths and limitations. These visualizations offered a comprehensive understanding of the role of Generative AI tools in software testing education, emphasizing their complementary strengths and overall utility.

D. DATA SCHEMA AND ALGORITHMS FOR PROCESSING AND ANALYZING STUDENT RESPONSES

The data collected for this study was stored in an SQLite database with two tables: ChatGPT and Copilot. Both tables followed the same schema, with the primary difference being the number of columns representing student responses. Table 3 summarizes the structure of the anonymized dataset. The dataset was initially prepared in a tabular format using Excel, with rows representing questions and columns capturing individual student responses. This simple structure allowed for seamless integration into the SQLite database, ensuring consistency between the ChatGPT and Copilot tables. By leveraging this approach, the workflow remained accessible and adaptable, requiring minimal technical overhead for dataset preparation and upload.

This study involved analyzing student assignments and their responses to a series of 11 questions. The responses were anonymized, and no specific student identifiers were included, thereby removing the need for individual consent. Approval for this classroom-based study was obtained from the Fanshawe College Research Ethics Board.

Algorithm 1 outlines the process used to collect, organize, and analyze student responses stored in the database. Next, the analysis approach was further detailed in algorithm 2, algorithm 3, and algorithm 4 outlining the generalized framework for analyzing responses for a specific Question and generating a comparative bar graph based on count-based analysis, thematic analysis, and, sentiment analysis respectively. The output is saved with a filename in the format `Question<Question_ID>.png`.

Algorithm 1 Algorithm for Processing and Analyzing Student Responses on Preparatory Testing Artifacts Generation

```
1:  Step 1: Extract responses from submitted assignments.
2:  Step 2: Anonymize the dataset to ensure privacy.
3:  Step 3: Structure the dataset as follows:
4:      Each column corresponds to a single Question's responses.
5:      Save the dataset in an Excel file (.xlsx) with two worksheets:
6:          (a) ChatGPT responses.
7:          (b) Copilot responses.
8:  Step 4: Export the dataset into an SQLite database.
9:      Create two tables in the database:
10:         (a) ChatGPT.
11:         (b) Copilot.
12: Step 5: Perform preliminary analysis and categorize the evaluation approach.
13: (a) Count-Based Approach:
14:     For free-response questions with suggested options, multiple-choice, or Likert scale questions that require justification:
15:         Extract responses column-by-column for each row.
16:         Match keywords in responses.
17:         Count occurrences of each keyword.
18: (b) Thematic Analysis:
19:     For open-ended, context-driven questions:
20:         Define thematic categories.
21:         Map keywords to each category.
22:         Count keyword mappings for each category.
23: (c) Sentiment Analysis:
24:     For open-ended questions:
25:         Apply a sentiment analysis model to the extracted text.
26:         Classify sentiment as positive, neutral, or negative based on predefined thresholds.
27: Step 6: Visualization.
28:     Since the same questions were given to both groups:
29:         Plot side-by-side bar graphs for comparative analysis of ChatGPT and Copilot responses.
30:         Use:
31:             X-axis: Question IDs or categories.
32:             Y-axis: Count, thematic mappings, or sentiment scores.
```

IV. AUTHORS' UTILIZATION OF PROMPT ENGINEERING TECHNIQUE FOR GENERATING TESTING PREPARATORY ARTIFACTS

Since this exercise involved crafting prompts, the authors experimented with prompt techniques to get the best results.

TABLE 3. Data schema of anonymized student responses.

Column Name	Column Description	Type
Question_ID	Unique identifier of each Question (1 to 11). This field is the primary key of this table.	Integer
Question _Description	Text content of each Question	Text
Student 1 Response	Response to this Question from the first student	Text
Student 2 Response	Response to this Question from the second student	Text
Student n Response	The same technique applies for recording responses from the remaining students where there is a new column for each additional student There were 26 students selected for ChatGPT and 21 for Copilot	Text

Algorithm 2 Count-Based Analysis for Comparing Question Response

```
1:  Step 1: Connect to the database.
2:    Establish a connection to the SQLite database file (DB_FILE).
3:  Step 2: Retrieve the row for the specified Question.
4:    Match the Question_ID to retrieve the row containing:
5:      (a) Question_ID.
6:      (b) Question_Description.
7:      (c) Responses from students (Student_Responses).
8:  Step 3: Initialize keyword counts.
9:    Define a list of keywords (KEYWORDS) based on the Question options.
10:   Initialize a count dictionary for each keyword.
11: Step 4: Analyze responses.
12: for each response in the row (excluding metadata columns like Question_ID and Description) do
13:     for each keyword in KEYWORDS do
14:         if the keyword is found in the response then
15:             Increment the corresponding keyword's count.
16:             if STOP_AFTER_FIRST_MATCH is True then
17:                 Stop searching for other keywords in this response.
18:             end if
19:         end if
20:     end for
21: end for
22: Step 5: Generate the visualization.
23:   Format the title of the bar graph as Question_ID.Question_Description.
24:   Plot the bar graph:
25:     X-axis: Keywords.
26:     Y-axis: Counts of responses for each keyword.
27:   Save the bar graph as Question<Question_ID>.png.
28: Step 6: Return results.
29:   Return the bar graph file and keyword counts.
```

The use of refinement of prompts in an interactive approach appeared to provide the best outcome. Here are the steps that were applied:

Algorithm 3 Thematic Analysis for Comparing Question Responses

```
1:  Step 1: Connect to the database.
2:    Establish a connection to the SQLite database file (DB_FILE).
3:  Step 2: Retrieve the row for the specified Question.
4:    Match the Question_ID to retrieve the row containing:
5:      (a) Question_ID.
6:      (b) Question_Description.
7:      (c) Responses from students (Student_Responses).
8:  Step 3: Define themes and keywords.
9:    Create a dictionary of themes with associated keywords:
10:     AI as a Supplement: {'supplement', 'complement'},
11:     Human Oversight: {'human', 'expertise'}, etc.
12: Step 4: Initialize theme counts.
13:   Set zero counts for each theme and an additional 'None' category.
14: Step 5: Analyze responses.
15: for each response (excluding metadata columns like Question_ID and Description) do
16:     if response is valid (not NaN) then
17:         Preprocess text (e.g., convert to lowercase, strip whitespace).
18:         for each theme do
19:             if any theme keyword appears in response then
20:                 Increment theme count.
21:                 Break.
22:             end if
23:         end for
24:         if no match found then
25:             Increment 'None' count.
26:         end if
27:     end if
28: end for
29: Step 6: Generate visualization.
30:   Format the title of the bar graph as Question_ID.Question_Description.
31:   Plot bar graph with:
32:     X-axis: Themes, Y-axis: Response counts.
33:   Save as Question<Question_ID>.png.
34: Step 7: Return results.
35:   Return graph and theme counts.
```

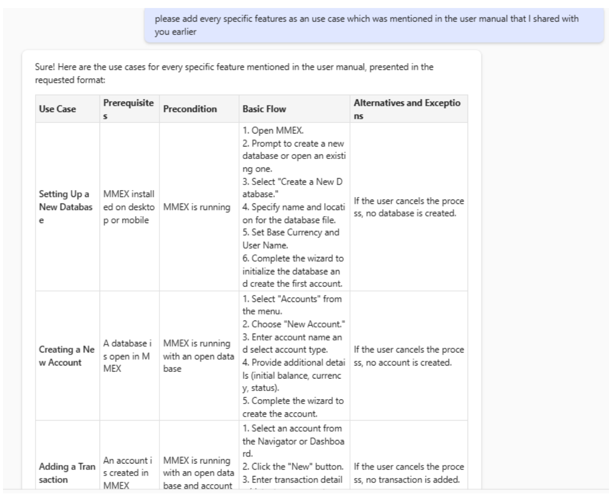
Algorithm 4 Sentiment Analysis for Comparing Question Responses

```
1:  Step 1: Connect to the database.
2:   Establish a connection to the SQLite database file (DB_FILE).
3:  Step 2: Retrieve the row for the specified Question.
4:   Match the Question_ID to retrieve the row containing:
5:     (a) Question_ID.
6:     (b) Question_Description.
7:     (c) Responses from students (Student_Responses).
8:  Step 3: Initialize sentiment counts.
9:   Define sentiment categories: Positive, Neutral, and Negative.
10:  Initialize a count dictionary for each sentiment category.
11: Step 4: Analyze responses.
12: for each response in the row (excluding metadata columns like Question_ID and Description) do
13:     Compute polarity scores using the VADER sentiment analyzer.
14:     if compound score > 0.05 then
15:         Classify the response as Positive.
16:         Increment the count for Positive.
17:     else if compound score < -0.05 then
18:         Classify the response as Negative.
19:         Increment the count for Negative.
20:     else
21:         Classify the response as Neutral.
22:         Increment the count for Neutral.
23:     end if
24: end for
25: Step 5: Generate the visualization.
26:    Format the title of the bar graph as Question_ID.Question_Description.
27:  Plot the bar graph:
28:     X-axis: Sentiment categories (Positive, Neutral, Negative).
29:    Y-axis: Counts of responses for each sentiment.
30:  Bars: Side-by-side comparison of counts for the two tools (e.g., ChatGPT and Copilot).
31:      Save the bar graph as Question<Question_ID>.png.
32: Step 6: Return results.
33:  Return the bar graph file and sentiment counts.
```

- Starting with a board query by asking generation questions to the LLM tools to establish the context and the scope of the needs.
- Progressively providing more detailed and specific instructions to refine the output.
- Clarifying and expanding the requirements to ensure comprehensive coverage.
- Providing a set of instructions for the specific format of the outputs.

The steps for constructing the prompts in the Travel Application were discussed in the earlier study [14]. As a result, this section describes how Copilot prompts were constructed whereas the same strategy can be applied in ChatGPT. The first prompt for Copilot included the Question of whether this tool has direct access to the GitHub repository of the MoneyManagerEx application. Once the Copilot responded that no direct access was available, progressively more detailed and specific instructions were provided to refine the output. The content from the user manual was crafted step by step, and incremental adjustments based on the responses were provided. Afterward, the requirements were clarified and expanded on them as needed to ensure comprehensive coverage of all aspects was observed. The LLM tool was asked to provide instruction in a structured tabular format to meet the needs effectively.

A subset of the use cases generated after the prompt was tuned further after sharing the user manual has been shown in Figure 3. This is followed by subsets of RTM in Figure 4, Test Cases in Figure 5, and detailed Test Scripts in Figure 6.



please add every specific features as an use case which was mentioned in the user manual that I shared with you earlier

Sure! Here are the use cases for every specific feature mentioned in the user manual, presented in the requested format:

Use Case	Prerequisites	Precondition	Basic Flow	Alternatives and Exceptions
Setting Up a New Database	MMEX installed on desktop or mobile	MMEX is running	1. Open MMEX. 2. Prompt to create a new database or open an existing one. 3. Select "Create a New Database." 4. Specify name and location for the database file. 5. Set Base Currency and User Name. 6. Complete the wizard to initialize the database and create the first account.	If the user cancels the process, no database is created.
Creating a New Account	A database is open in MMEX	MMEX is running with an open database	1. Select "Accounts" from the menu. 2. Choose "New Account." 3. Enter account name and select account type. 4. Provide additional details (initial balance, currency, status). 5. Complete the wizard to create the account.	If the user cancels the process, no account is created.
Adding a Transaction	An account is created in MMEX	MMEX is running with an open database and account	1. Select an account from the Navigator or Dashboard. 2. Click the "New" button. 3. Enter transaction detail	If the user cancels the process, no transaction is added.

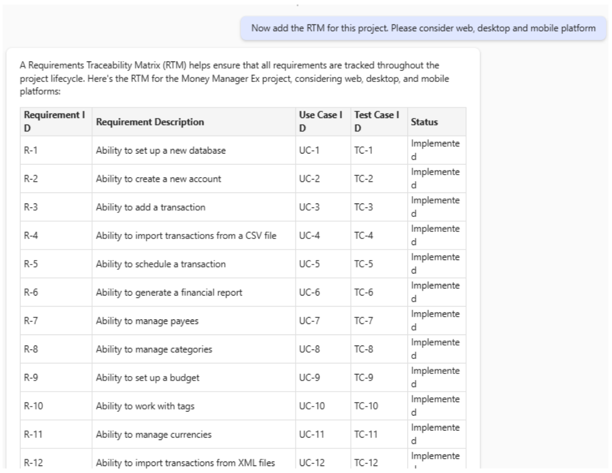
FIGURE 3. A subset of use cases generated after sharing the user manual in the prompt in Copilot.

V. RESULTS

A. EVALUATION OF RESULTS FROM COUNT-BASED APPROACH

Figure 7 and Figure 8 illustrate the results of responses analyzed from four different questions using a count-based approach after providing the keyword matching, comparing student experiences with ChatGPT-3.5 and Copilot. The bar graphs present a side-by-side analysis of the findings, highlighting key differences.

In Figure 7a, 66.67% of students responded positively ('Yes') about Copilot, compared to 53.85% for ChatGPT-3.5. Additionally, while 33.33% of students responded 'No' for Copilot, this figure was slightly higher for ChatGPT-3.5 at 42.31%, with 3.85% of students selecting 'Yes and No'. Although students generally showed more positive experiences with Copilot, across both studies, the

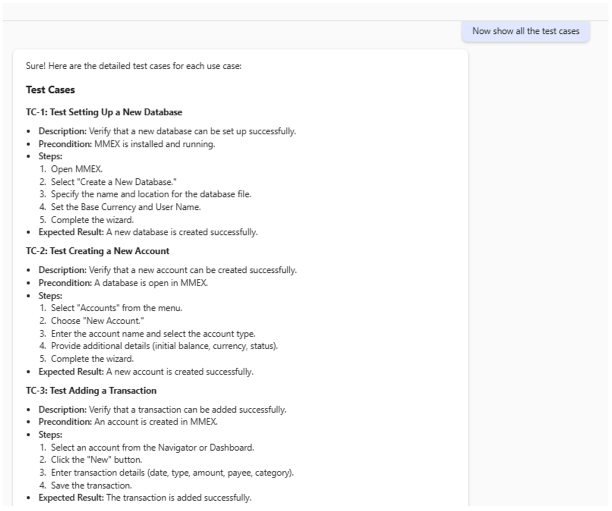


Now add the RTM for this project. Please consider web, desktop and mobile platform

A Requirements Traceability Matrix (RTM) helps ensure that all requirements are tracked throughout the project lifecycle. Here's the RTM for the Money Manager Ex project, considering web, desktop, and mobile platforms:

Requirement ID	Requirement Description	Use Case ID	Test Case ID	Status
R-1	Ability to set up a new database	UC-1	TC-1	Implemented
R-2	Ability to create a new account	UC-2	TC-2	Implemented
R-3	Ability to add a transaction	UC-3	TC-3	Implemented
R-4	Ability to import transactions from a CSV file	UC-4	TC-4	Implemented
R-5	Ability to schedule a transaction	UC-5	TC-5	Implemented
R-6	Ability to generate a financial report	UC-6	TC-6	Implemented
R-7	Ability to manage payees	UC-7	TC-7	Implemented
R-8	Ability to manage categories	UC-8	TC-8	Implemented
R-9	Ability to set up a budget	UC-9	TC-9	Implemented
R-10	Ability to work with tags	UC-10	TC-10	Implemented
R-11	Ability to manage currencies	UC-11	TC-11	Implemented
R-12	Ability to import transactions from XML files	UC-12	TC-12	Implemente

FIGURE 4. A subset of RTM generated after sharing the user manual in the prompt in Copilot.



Now show all the test cases

Sure! Here are the detailed test cases for each use case:

Test Cases

TC-1: Test Setting Up a New Database

- **Description:** Verify that a new database can be set up successfully.
- **Precondition:** MMEX is installed and running.
- **Steps:**
  1. Open MMEX.
  2. Select "Create a New Database."
  3. Specify the name and location for the database file.
  4. Set the Base Currency and User Name.
  5. Complete the wizard.
- **Expected Result:** A new database is created successfully.

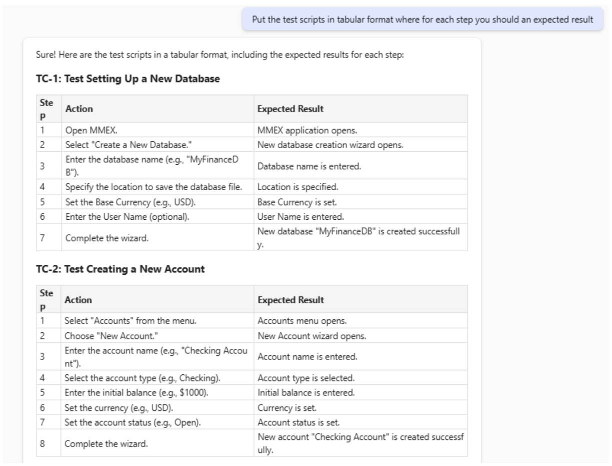
TC-2: Test Creating a New Account

- **Description:** Verify that a new account can be created successfully.
- **Precondition:** A database is open in MMEX.
- **Steps:**
  1. Select "Accounts" from the menu.
  2. Choose "New Account."
  3. Enter the account name and select the account type.
  4. Provide additional details (initial balance, currency, status).
  5. Complete the wizard.
- **Expected Result:** A new account is created successfully.

TC-3: Test Adding a Transaction

- **Description:** Verify that a transaction can be added successfully.
- **Precondition:** An account is created in MMEX.
- **Steps:**
  1. Select an account from the Navigator or Dashboard.
  2. Click the "New" button.
  3. Enter transaction details (date, type, amount, payee, category).
  4. Save the transaction.
- **Expected Result:** The transaction is added successfully.

FIGURE 5. A subset of test cases generated after crafting the prompt in Copilot.



Put the test scripts in tabular format where for each step you should an expected result

Sure! Here are the test scripts in a tabular format, including the expected results for each step:

TC-1: Test Setting Up a New Database

Step	Action	Expected Result
1	Open MMEX.	MMEX application opens.
2	Select "Create a New Database."	New database creation wizard opens.
3	Enter the database name (e.g., "MyFinanceDB").	Database name is entered.
4	Specify the location to save the database file.	Location is specified.
5	Set the Base Currency (e.g., USD).	Base Currency is set.
6	Enter the User Name (optional).	User Name is entered.
7	Complete the wizard.	New database "MyFinanceDB" is created successfully.

TC-2: Test Creating a New Account

Step	Action	Expected Result
1	Select "Accounts" from the menu.	Accounts menu opens.
2	Choose "New Account."	New Account wizard opens.
3	Enter the account name (e.g., "Checking Account").	Account name is entered.
4	Select the account type (e.g., Checking).	Account type is selected.
5	Enter the initial balance (e.g., $1000).	Initial balance is entered.
6	Set the currency (e.g., USD).	Currency is set.
7	Set the account status (e.g., Open).	Account status is set.
8	Complete the wizard.	New account "Checking Account" is created successfully.

FIGURE 6. A subset of generated test scripts after crafting the prompt in Copilot.

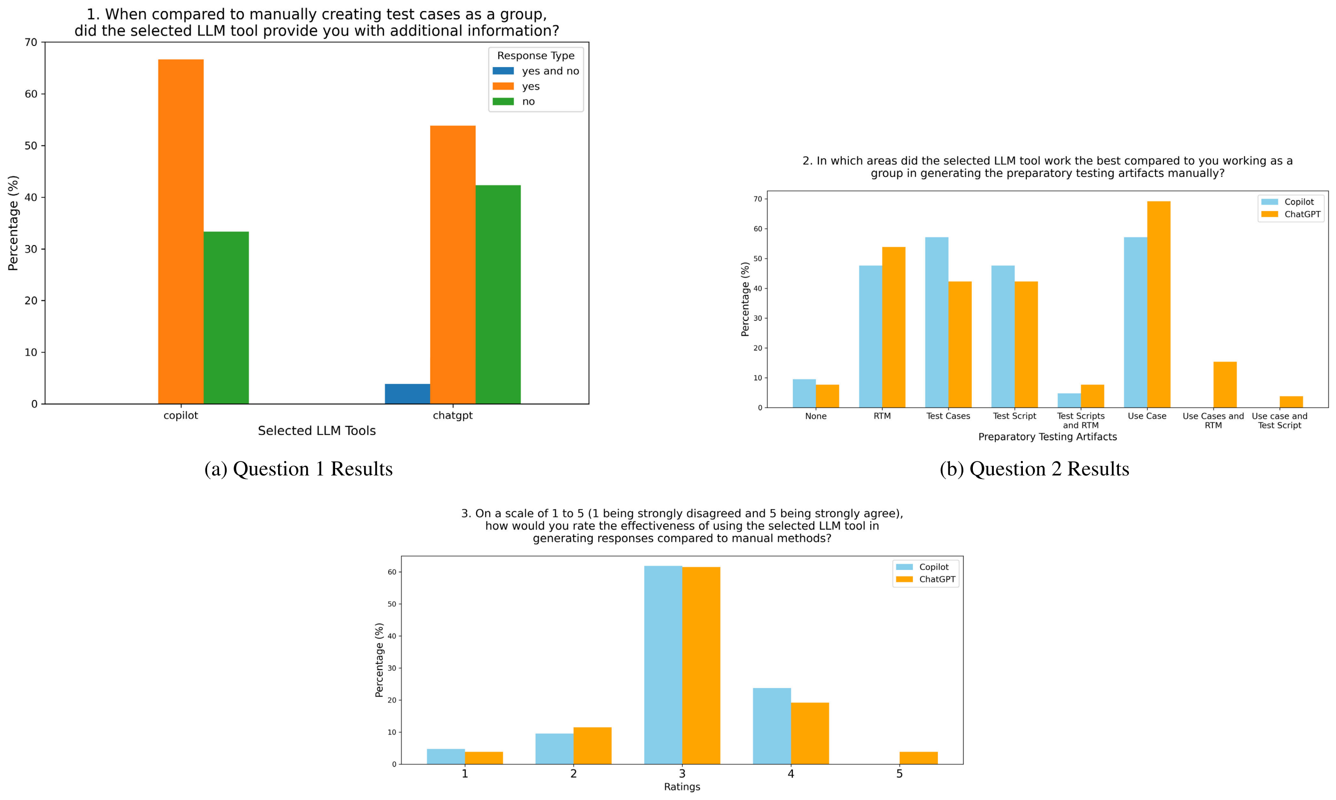
majority of students acknowledged that integrating AI tools provided them with additional valuable information. This reinforces the potential of these LLM tools in enhancing learning and testing processes.

Figure 7b compares student responses on the effectiveness of ChatGPT and Copilot in assisting with generating testing artifacts. The analysis reveals that Use Cases was the most preferred area for both tools, with 69.23% of students identifying ChatGPT as most effective in this area and 57.14% favoring Copilot. For ChatGPT, RTM was the second most preferred area, scoring 53.85%, while for Copilot, Test Cases emerged as the second preference, with 57.14%. The third preference differed between the tools with Test Cases for ChatGPT scoring 42.31% and RTM for Copilot was 47.62%. Interestingly, for ChatGPT, several students mentioned combined categories such as 'Use Cases and RTM', with 15.38% of students preferring it. Only a small percentage of students indicated no improvement with either tool, with 7.69% for ChatGPT and 9.52% for Copilot. Overall, the findings highlight that Use Cases were the most effective and preferred testing area for both tools, while other generated artifacts of RTM, Test cases, and Test Scripts also showed notable confidence.

Figure 7c demonstrates that both ChatGPT and Copilot are largely perceived as moderately effective in generating responses compared to manual methods, with most participants assigning a neutral rating of 3 (61.90% for Copilot and 61.54% for ChatGPT). While dissatisfaction levels are similarly low for both tools (14.28% for Copilot and 15.39% for ChatGPT with ratings of below 3), Copilot slightly edges out ChatGPT in positive perceptions, with 23.08% of students rating it as effective or strongly effective (4 or 5), compared to 22.22% for ChatGPT in the same range. ChatGPT also received one 'strongly agree' rating (5), which was absent for Copilot. These results suggest that while both tools are generally viewed as moderately useful, Copilot may have a marginal advantage in student satisfaction.

The findings from Figure 8a highlight differences in the aspects identified as missing between preparatory testing artifacts generated by ChatGPT and those supported by Copilot when compared to manually created artifacts. For Question 4, keywords were mapped to categories such as 'Non-Functional', 'Features', and 'Test Steps'. The keyword 'Features' refers to test cases missed related to application functionalities, representing gaps in the students' manually created artifact submissions. Both tools demonstrated a strong focus on non-functional testing aspects, either explicitly or through specific attributes.

The keyword 'Non-Functional' explicitly accounted for 23.08% of responses for ChatGPT and 28.57% for Copilot. Additionally, students identified specific non-functional attributes such as usability, scalability, performance, privacy, and data protection, all of which fall under the umbrella of non-functional testing. ChatGPT responses emphasized gaps related to features (15.38%), details (15.38%), and data (15.38%), indicating a broader representation of missing aspects. In contrast, Copilot focused more heavily on performance (23.81%) and features (28.57%). Notably, more



(a) Question 1 Results

(b) Question 2 Results

(c) Question 3 Results

FIGURE 7. Aggregated results for Questions 1, 2, 3 where the count-based approach was used for evaluation.

ChatGPT users reported no missing aspects ('None' at 30.77%) compared to Copilot users (19.05%). These findings suggest that both tools effectively identify non-functional testing gaps, but they differ in their representation, with ChatGPT highlighting broader aspects like features and details, while Copilot focuses more specifically on performance and feature-related deficiencies.

Figure 8b highlights the distinct strengths of Copilot and ChatGPT in assisting with testing tasks. Copilot's standout feature was 'Test Case Generation Speed', with an overwhelming 85% of students identifying it as a key advantage, emphasizing its efficiency in generating test cases quickly. A smaller percentage (10%) appreciated its Ease of Use, suggesting that while Copilot is primarily valued for its speed, some students found it user-friendly as well. However, other combined aspects, such as Test Case Accuracy and Test Case Coverage' or 'Test Case Generation Speed and Ease of Use' received no preference, indicating that these combinations were not impactful for students when using Copilot.

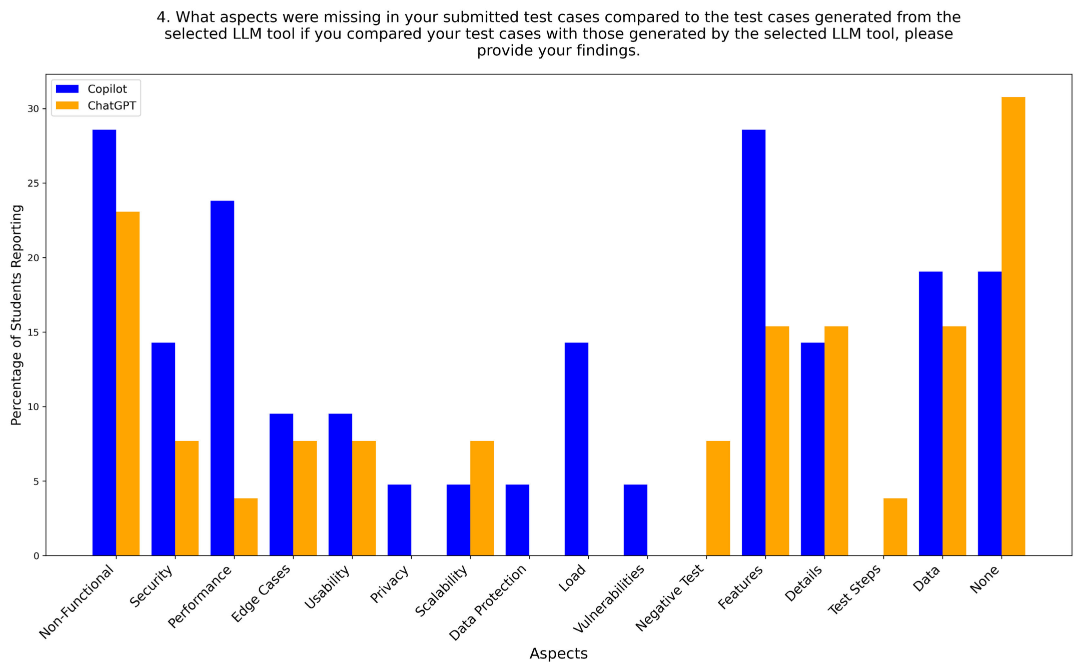
In contrast, ChatGPT demonstrated a more balanced distribution of strengths. While 'Test Case Generation Speed' was also its most recognized feature (53.85%), this was about two-thirds of Copilot's performance in the same category. Notably, ChatGPT had a strong showing in 'Ease of Use', with 15.38% of students highlighting this aspect. Additionally, ChatGPT outperformed Copilot in 'Test Case Coverage' (11.54% versus 5%) and received some recognition for combined aspects like 'Test Case Generation Speed and Ease of Use' (7.69%) and 'Test Case Accuracy and Test Case Coverage' (3.85%).

Overall, these findings reveal that Copilot excels in speed, making it the preferred tool for quick test generation, while ChatGPT offers more versatile performance, balancing ease of use and broader coverage. This indicates that the two tools have complementary strengths to assist with preparatory software testing artifact generation.

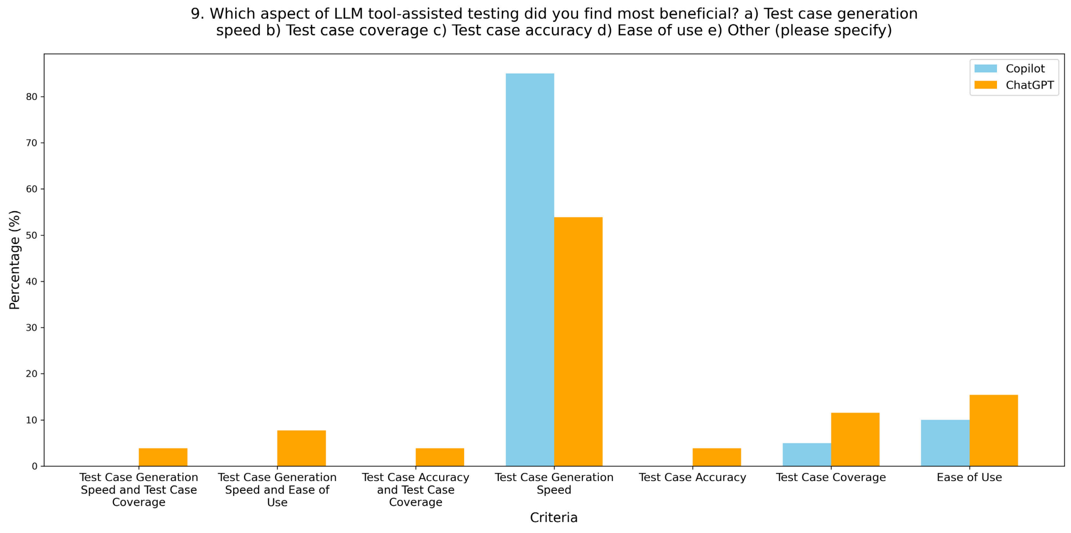
B. USING SENTIMENT ANALYSIS TO EVALUATE THE RESULTS

Figure 9 presents the sentiment analysis of student responses to the assigned questions. These open-ended questions were designed to gauge students' perceptions of utilizing LLM tools for generating preparatory testing artifacts. Sentiment analysis provides valuable insights into whether these tools effectively meet student needs and whether they should be incorporated into software testing courses to enhance learning outcomes.

Figure 9a illustrates aggregated responses to Question 6, which asked whether tools like ChatGPT and Copilot could be beneficial additions to the testing process. Most responses across both cohorts were positive, with



(a) Question 4 Results



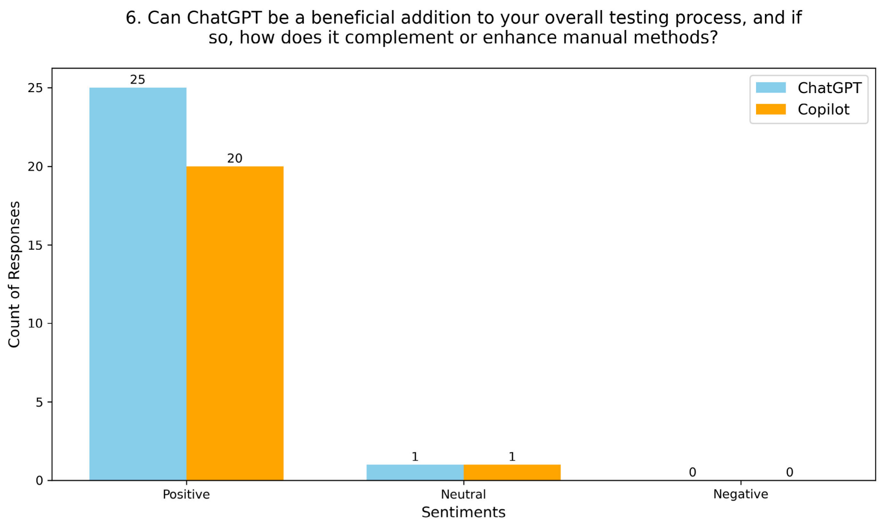
(b) Question 9 Results

FIGURE 8. Aggregated results for Questions 4 and 9 where the count-based approach was used for evaluation.

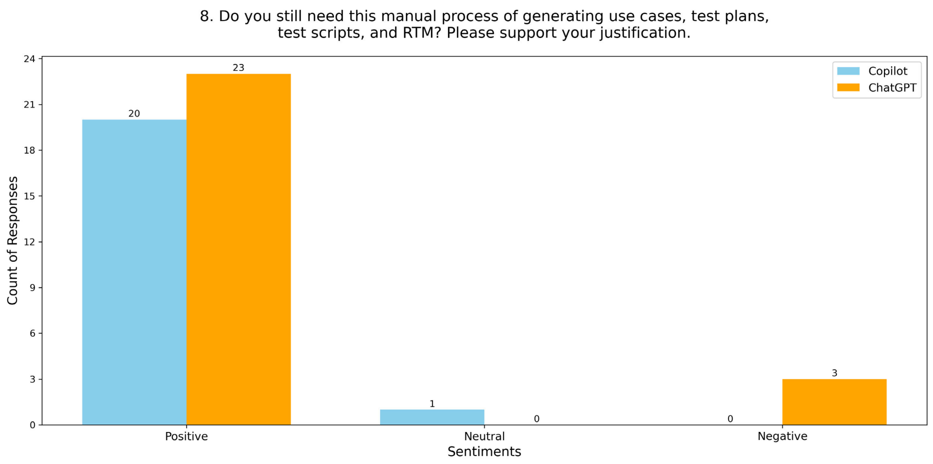
only one neutral sentiment and no negative responses. This overwhelmingly positive sentiment underscores the perceived value of integrating these tools into testing practices.

Figure 9b depicts student opinions on whether the manual process of generating preparatory testing artifacts is still necessary. ChatGPT and Copilot both elicited high percentages of positive responses, with 88.46% (23 out of 26 submissions) for ChatGPT and 95.24% (20 out of 21 submissions) for Copilot. Neutral responses accounted for 11.54% (3 out of 26) for ChatGPT and 4.76% (1 out of 21) for Copilot, while negative responses were nonexistent for both tools. These results suggest that while LLM tools are highly appreciated, students still recognize the importance of manual processes in specific contexts.

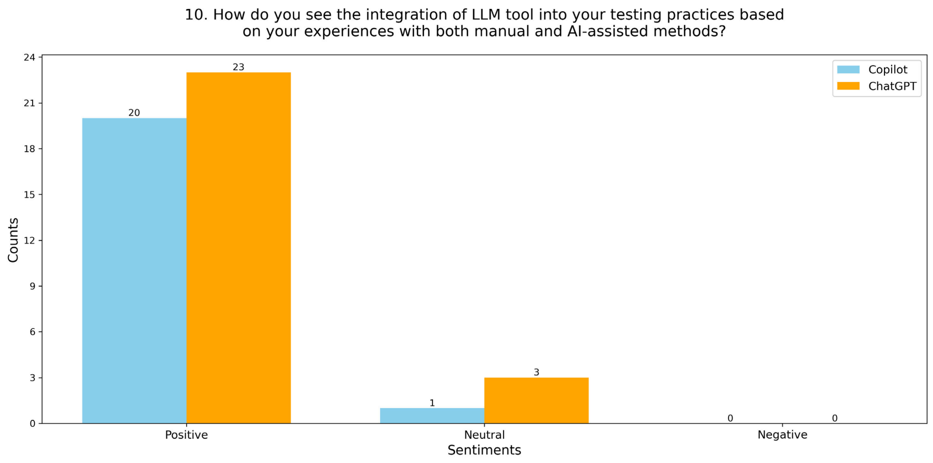
Figure 9c summarizes responses to Question 10, which asked students about their views on integrating ChatGPT or Copilot into testing practices. The results show strong positive feedback for both tools. Copilot received 20 positive responses out of 21 (95.24%), while ChatGPT had 23 positive responses out of 26 (88.46%). Neutral feedback was minimal, with 1 response (4.76%) for Copilot and 3 responses (11.54%) for ChatGPT. No negative feedback was recorded for either tool.



(a) Question 6 Results



(b) Question 8 Results



(c) Question 10 Results

FIGURE 9. **Aggregated results for Questions 6, 8, and 10, analyzed using sentiment analysis.**

These results highlight that both LLM tools are seen as valuable additions to testing practices. Copilot's slightly higher percentage of positive responses might be because it was used with a well-documented open-source project, which helped students create more effective prompts. On the other hand, ChatGPT's responses may have been influenced by its more general-purpose design. This suggests that providing clear documentation and resources can improve the use of LLM tools, although further research is needed to confirm this in different scenarios.

While the findings suggest that both tools have their strengths, the limited data prevents drawing definitive conclusions. Copilot appeared to create a stronger positive impression, potentially due to its use with well-documented open-source material, while ChatGPT showed consistent support across varied scenarios. These results indicate the potential of both tools to complement each other in enhancing the software testing process, but further studies are needed to validate these observations.

C. ANALYSIS USING THEMATIC APPROACH

Figure 10 demonstrates the result of when thematic analysis was employed to categorize and interpret responses to open-ended questions. The process involved extracting specific rows from a database and matching keywords in the responses to predefined thematic categories. This method facilitated the identification of patterns and insights within the qualitative data, providing a structured framework for interpretation.

As shown in Figure 10a, for Question 5, responses were categorized into themes that highlighted the feasibility and applicability of test cases. The following keywords were mapped to the corresponding themes:

- will not be needed, do not require, did not seem useful, it is not required, falls in this category → Not Required for this application
- may not be applicable, not familiar, incorrectly identified, irrelevant, wrong information, not part of, search for places → Irrelevant Features
- nonfunctional, non-functional → Nonfunctional Testing Resource Constraint
- cases will not apply, beyond scope, out of scope, not applicable, assumed functionality, some are not completely → Out of Scope
- functions are not there, is not found, search for places, I don't think is feasible, not be feasible, missing, not available, not included in the module, lacks, not implemented, not present, not feasible → Features Not Available
- none that does not fall into this category, generated all the correct test cases, feasible, helpful, useful, applicable, provides direction → Feasible or None
- details → Lacking Details to Implement
- test cases require special software, not feasible, time constraints, limited resources, specialized hardware, cannot test → Testing Resource Constraints

The analysis revealed that both ChatGPT and Copilot occasionally generate test cases that are not feasible or applicable. For Copilot, the most common issue was creating test cases for features not available (11 instances), despite having access to structured resources like user manuals, which likely contributed to its ability to produce some feasible test cases (4 instances). ChatGPT also struggled with unavailable features (8 instances) but outperformed Copilot in generating feasible test cases (8 instances compared to Copilot's 4). These results highlight that while access to user manuals may help align outputs in some cases, both tools still show a tendency to hallucinate features or misinterpret application boundaries, emphasizing the need for clearer input and human oversight during use.

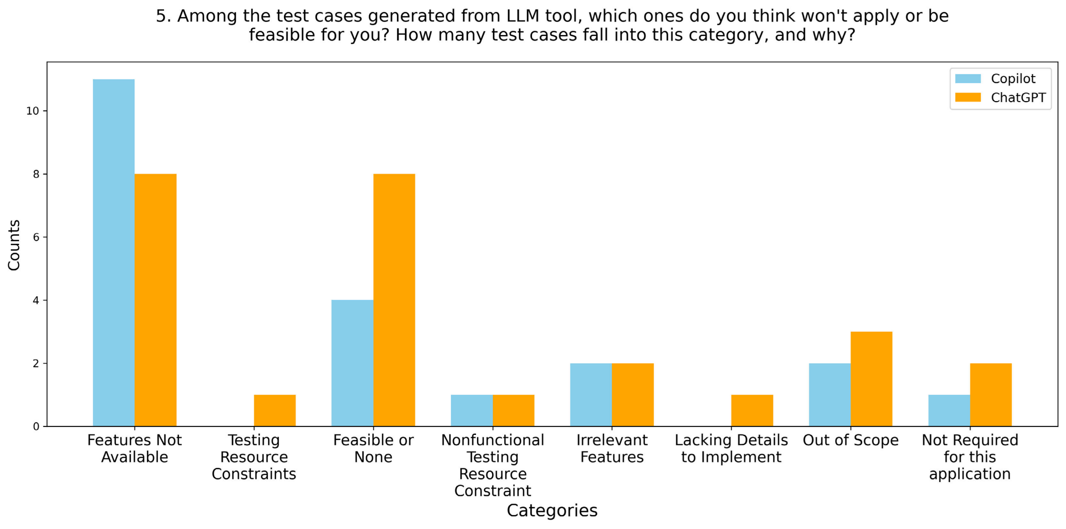
Figure 10b shows responses to Question 7, which examines what students identified as missing in their submitted test cases compared to the generated test cases from LLMs. The responses were analyzed by categorizing keywords into themes related to different types of testing. The keywords mapped to the corresponding themes are as follows:

- non-functional → Non-functional tests
- performance → Performance tests
- security → Security tests
- compatibility → Compatibility tests
- edge case, boundary → Edge cases
- usability, accessibility → Usability/accessibility
- import, export, data → Data
- additional test cases, in the area functional, coverage for buttons → Functional tests
- covered more, comprehensive → Coverage comparison
- steps to reproduce, description → Additional details/ structure
-

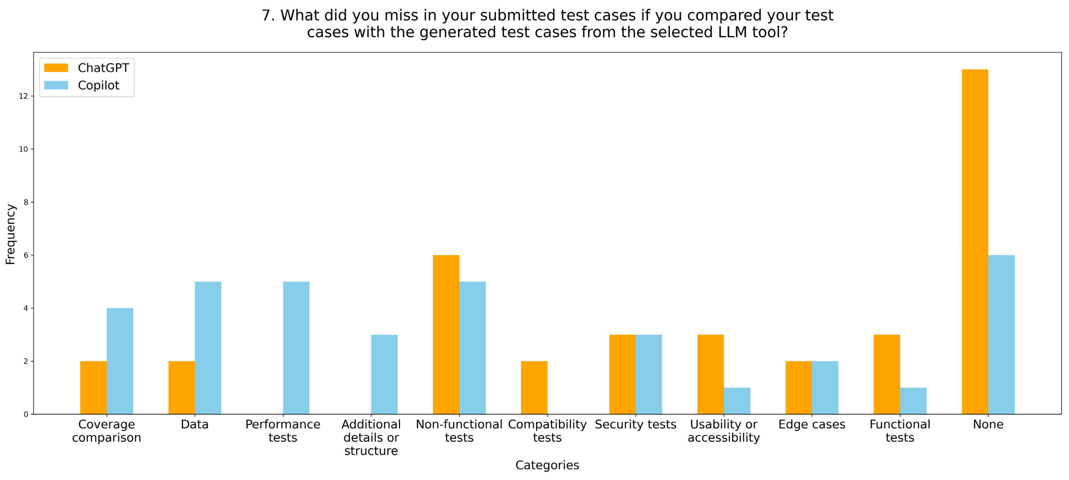
The thematic analysis of Question 7 shown in Figure 10b provides insights into how participants perceived ChatGPT and Copilot in the context of testing-related tasks. The findings reveal notable differences and similarities in the tools' perceived strengths and areas of focus.

For ChatGPT, the most frequently mentioned category was Non-functional Tests (6 mentions), indicating that students found these types of test cases lacking in their submissions compared to ChatGPT's outputs. Categories like Security Tests (3 mentions), Usability/Accessibility (3 mentions), and Functional Tests (3 mentions) also stood out, highlighting ChatGPT's perceived ability to address user-centric and functional aspects. Other themes, such as Edge Cases (2 mentions), Data Management (2 mentions), and Coverage Comparison (2 mentions), were less frequently identified but still noteworthy. A significant portion of responses (13 mentions) fell into the 'None' category, suggesting that many participants did not perceive specific types of missed test cases when comparing their work to ChatGPT's outputs.

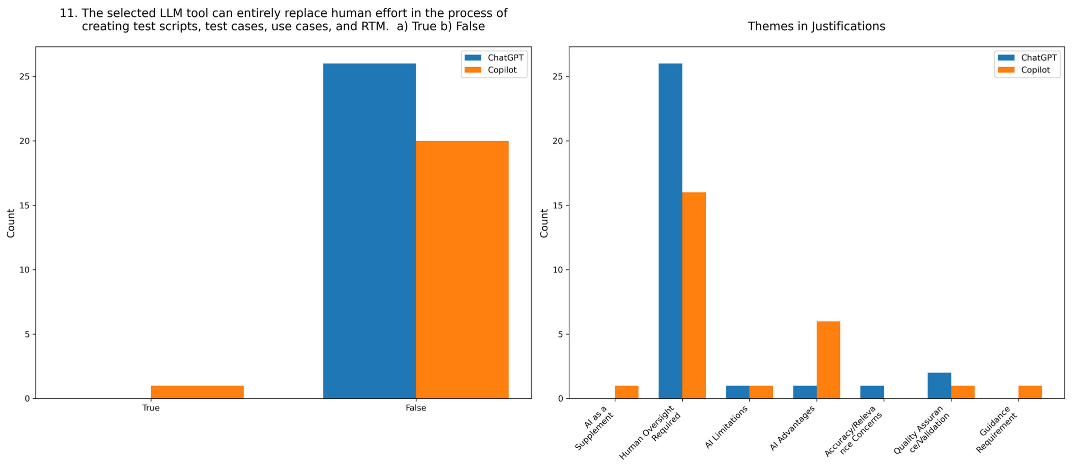
For Copilot, the most frequently mentioned categories were Data (5 mentions) where students missed adding data input in their test scripts, and Performance Tests (5 mentions), reflecting its perceived strength in handling data-related and performance-focused test cases. Non-functional Tests (5 mentions) and Coverage Comparison (4 mentions) were also notable, highlighting Copilot's ability to identify gaps



(a) Question 5 Results



(b) Question 7 Results



(c) Question 11 Results

FIGURE 10. Results obtained from Questions 5, 7, and 11 where the thematic approach was applied.

and provide broader test coverage. Categories like Security Tests (3 mentions), Edge Cases (2 mentions), and Functional Tests (1 mention) were less common, suggesting fewer overlaps with these types of test cases. Additionally, Copilot received 3 mentions for Additional Details/Structure, a unique category that reflects its ability to generate more structured and actionable test cases. Only 6 responses fell into the 'None' category, indicating that Copilot's outputs were more frequently seen as supplementing students' submitted test cases.

Overall, the findings suggest that students most frequently missed non-functional test cases in their submissions, as highlighted by both tools' outputs. ChatGPT demonstrated strengths in generating a variety of non-functional test cases, including usability, security, and performance testing, reflecting its broader analytical capabilities. Copilot also addressed non-functional testing effectively but stood out by generating test cases with detailed data inputs, which is a critical aspect of comprehensive testing. Both tools showed comparable utility in areas like security testing and edge case identification, though neither tool dominated these categories.

Question 11 examines students' views on whether AI tools like ChatGPT and Copilot can replace human effort in software testing tasks. This question had two parts where at first, students were asked if they think AI can replace humans in generating these preparatory testing tasks (answered with a 'yes' or 'no'), and second, they were asked to provide justifications for their responses. Figure 10c summarizes the results of this question using two different approaches. The left-hand figure presents a count-based analysis of the 'yes' or 'no' responses. In contrast, the right-hand figure categorizes the justifications into thematic groups based on the following keywords:

- supplement, complement → AI as a Supplement
- human, expertise → Human Oversight Required
- limitations, lacks context, missing details, wrong information, hallucinate → AI Limitations
- efficient, speed → AI Advantages
- not accurate, not relevant, not in context → Accuracy/Relevance Concerns
- understanding issues, misunderstanding → Understanding Concerns
- quality assurance, validation required, human validation → Quality Assurance/Validation
- guidance needs prompting, user guidance → Guidance Requirement

For ChatGPT, all 26 responses indicated False, showing strong agreement that ChatGPT cannot fully replace human involvement. This reflects a strong skepticism about ChatGPT's current capabilities in independently handling tasks like creating test scripts, use cases, and related outputs.

For Copilot, 20 responses were marked as False, while only 1 response indicated True. Although this suggests slightly more confidence in Copilot compared to ChatGPT, most participants believe that Copilot, like ChatGPT, cannot replace human effort entirely.

When examining themes, the most frequently mentioned category for both tools was Human Oversight Required. ChatGPT had 26 mentions, and Copilot had 16 mentions, emphasizing the necessity of human expertise to ensure accuracy, relevance, and quality. This underscores a shared belief that these tools are not yet self-sufficient.

Under AI Limitations, both ChatGPT and Copilot received only 1 mention each, indicating that participants acknowledge some constraints, such as lack of context or accuracy, but do not see these as dominant issues. Conversely, AI Advantages revealed a significant difference: Copilot had 6 mentions, whereas ChatGPT had just 1. This suggests that Copilot is perceived as offering more benefits, particularly in terms of efficiency and speed.

Concerns about Accuracy/Relevance were raised once for ChatGPT but not for Copilot, reflecting a minor worry about ChatGPT's ability to produce precise outputs. The Quality Assurance/Validation theme received 2 mentions for ChatGPT and 1 for Copilot, reinforcing the idea that human validation remains essential for both tools to ensure output quality.

The theme AI as a Supplement received just 1 mention for Copilot and none for ChatGPT, indicating minimal recognition of these tools as effective complements to human efforts. Finally, Guidance Requirements were not prominent themes with only a single mention of Copilot under Guidance Requirement, suggesting these were not significant concerns for participants. Since every response fell into categories into one of the given themes, this figure did not show a 'None' category.

These findings indicate that students do not believe AI tools like ChatGPT or Copilot can fully replace human effort in preparatory software testing tasks. However, they recognize the value these tools bring, particularly in faster test generation, creating RTMs, and developing test cases. While these tools have limitations, they can enhance manual testing efforts by providing a foundation for brainstorming and offering templates for comparison, ultimately empowering testers rather than replacing them.

VI. ANALYSIS AND DISCUSSION

This study builds upon prior research [14] that evaluated ChatGPT's role in generating preparatory testing artifacts for a proprietary travel application. The earlier study focused on two specific questions: which areas ChatGPT outperformed manual, group-generated artifacts, and which aspects of ChatGPT-assisted testing were most beneficial. That study involved 26 students from a single cohort, who used ChatGPT to generate testing artifacts for the custom Travel application. ChatGPT demonstrated strengths in test case generation speed and was particularly effective in generating Use Cases, followed by RTM and Test Cases.

A. COMPARATIVE ANALYSIS WITH PRIOR STUDY

This study revisits the same set of responses from the 26 students in the earlier research but expands the analysis to cover all 11 questions, including the original two, to allow for a more comprehensive understanding. Additionally, it introduces a comparative element by incorporating responses from a second cohort of 21 students who took the course in the subsequent semester. This second cohort used Copilot to generate testing artifacts for the MoneyManager application, a different but equally practical context with multiplatform capabilities, which made it a suitable candidate for software testing education.

When comparing findings from Question 2 in this study to the earlier work (Figure 7b), Use Cases remained the dominant artifact across tools. Similarly, as shown in Figure 8b, 'test generation speed' continued to be the most significant benefit observed for Copilot, mirroring ChatGPT's results.

While the earlier study did not analyze the remaining nine questions, this study expands the scope by examining responses to all 11 questions and incorporating Copilot for the first time. The analysis revealed that non-functional testing was a critical area where testers missed the most test cases in their manual submissions, along with the necessary data for those test cases. For both LLM tools, manual validation was required to address inaccuracies and ambiguities in the outputs.

This study also introduced a comparative element, with Copilot being exclusively used for open-source applications and ChatGPT applied to the custom web-based travel application. The earlier study noted concerns about sharing sensitive information when teaching with proprietary applications, as ChatGPT might potentially use such data for training. Using open-source applications mitigated this risk, as they inherently avoid issues related to sensitive information, offering a safer alternative for educational purposes.

B. RESPONSE TO RESEARCH QUESTIONS

The results obtained in this study address the research questions mentioned:

RQ1: Can a single teaching approach be effective for all testing types, or should different contexts have specialized methods?

The students applied various testing methodologies tailored to the applications they tested, despite differences in cohorts, semesters, tools, and testing contexts. The teaching approach remained consistent, introducing concepts through theoretical lectures, reinforcing understanding through group work and in-class exercises, and allowing independent exploration. Minor adjustments, such as the inclusion of multiplatform testing, ensured relevance across diverse contexts.

Despite these variations, students demonstrated the ability to critically analyze both manual and AI-generated artifacts and identified common strengths and limitations of tools when compared to the manual process of generating preparatory testing artifacts. For both ChatGPT and Copilot, students highlighted 'test case generation speed' as the most advantageous factor, alongside other benefits like 'data handling' and 'feature identification'. Notably, students testing open-source applications expressed higher satisfaction with Copilot, as its outputs better aligned with application requirements. However, this observation, based on limited data, warrants further validation with larger, more diverse samples. The exercise also revealed limitations, such as hallucinated test artifacts labeled as 'Features Not Available' underscoring the importance of human validation when integrating AI tools into testing practices. Additionally, this exercise gave students a practical understanding of how AI can be integrated into real-life testing solutions, bridging the gap between theoretical knowledge and real-world applications. This consistency across different settings validates the effectiveness of a universal teaching framework with minor contextual tweaks.

RQ2: What role should AI-based testing play in the curriculum, and how should it be implemented?

The study suggests that integrating AI-based testing into capstone projects, rather than as standalone courses, provides students with real-world exposure. This integration allows students to balance foundational manual testing techniques with AI-assisted approaches, fostering critical thinking skills. Comparative student responses revealed a moderate effectiveness rating of 3 (on a scale of 1 to 5) for both Copilot and ChatGPT, with Copilot generating artifacts that better aligned with project requirements. This indicates that while AI-based testing holds promise, it should be implemented as a complementary component, ensuring that students develop both technical and critical evaluation skills rather than relying solely on automated tools.

RQ3: Does incorporating LLMs into testing education help students develop skills that align with industry needs?

This study demonstrates that incorporating LLMs into testing education provides a balanced opportunity to develop both soft and technical skills. According to Hamid and Ikram [51], the industry prioritizes problem-solving, critical thinking, and technical skills in novice testers. Exercises involving open-source applications equipped students with practical, industry-relevant skills by bridging theoretical knowledge with real-world testing scenarios. However, the thematic analysis revealed limitations such as 'Out of Scope' functionalities in AI-generated outputs, emphasizing the importance of human oversight. Additionally, integrating AI into the curriculum in a way that complements foundational learning ensures students can critically assess and utilize AI tools, aligning their skills with industry demands.

VII. THREATS TO VALIDITY

The scope of this study was limited to two specific AI tools and a dataset collected from 47 students across two semesters within the context of a software testing course. While the findings provide valuable insights, they are influenced by the specificity of the dataset and the educational context in which it was collected. The tools used, ChatGPT and Copilot, represent popular subsets of LLM tools; however, other Generative AI technologies exist, and future work could extend this analysis to evaluate their effectiveness in similar contexts. Further research is needed to explore the long-term impacts of AI tool integration on student learning and performance. Expanding the analysis to include additional tools, larger datasets, and diverse application domains could enhance the generalizability of these results.

The evaluation strategy combined count-based, thematic, and sentiment analyses. Sentiment analysis utilized the VADER tool from the NLTK library, applied uniformly across all questions to standardize the framework and ensure reproducibility. While effective for general sentiment evaluation, VADER may not fully capture nuances specific to the software testing domain, potentially leading to gaps in context-specific sentiment interpretation. Count-based analysis systematically identified patterns but may have overlooked relevant keywords outside the predefined framework. Thematic analysis provided meaningful insights by grouping data into categories, but it relied on human interpretation, which can introduce subjectivity.

Despite these limitations, careful review was conducted to ensure no critical aspects were excluded from the study. The combination of these techniques offered a balanced approach, capturing both quantitative and qualitative insights. Future studies should consider integrating domain-specific sentiment analysis tools and expanding the scope to other Generative AI technologies to improve the granularity and comprehensiveness of AI tools' evaluation in software testing education.

VIII. CONCLUSION

This study demonstrates the potential of AI tools like ChatGPT and Copilot to enrich software testing education by bridging theoretical learning with practical, tool-driven exercises. Such integration prepares students for real-world challenges while fostering critical technical and analytical skills. The findings underscore the importance of complementing AI tools with human expertise to ensure the accuracy and relevance of testing artifacts.

The overall teaching approach in the course combined theoretical lectures, group activities, in-class exercises, and independent assignments, providing students with a well-rounded framework to explore how a software testing project is conducted within a real-world scenario. Within this structure, an exercise was designed to critically compare manual and AI-generated test artifacts, allowing students to evaluate the strengths and limitations of AI tools in practical settings. Spending the first 9 weeks creating manual artifacts enabled students to build a solid theoretical and practical foundation before evaluating AI-generated artifacts.

From this study, it was observed that open-source programs provided a slight advantage by offering more accessible information for crafting effective prompts for LLMs, particularly when using Copilot. While this highlights the potential benefits of structured and comprehensive input, further validation is required to confirm these findings across broader contexts.

To advance the pedagogy of software testing education, integrating AI tools into structured exercises offers an innovative approach to teaching emerging technologies. By emphasizing the iterative process of generating, evaluating, and refining artifacts, educators can help students develop the critical thinking and adaptability needed to navigate evolving industry demands.

ACKNOWLEDGMENT

The authors want to acknowledge Dr. Dev Sainani, the Associate Dean of the School of Information Technology, Fanshawe College, and the Fanshawe Research Ethical Board (REB) for their support towards this work.

DATA AVAILABILITY STATEMENT

The data analyzed in this study are institutional data and cannot be shared publicly due to confidentiality and institutional policies at this time. Requests for data access may be directed to the corresponding author but will be subject to institutional approval.

REFERENCES

[1] N. Alshahwan, M. Harman, and A. Marginean, "Software testing research challenges: An industrial perspective," in *Proc. IEEE Conf. Softw. Test., Verification Validation (ICST)*, Apr. 2023, pp. 1–10.

[2] H. Washizaki, Ed. (2024). *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0*. IEEE Computer Society. Accessed: Feb. 2025. [Online]. Available: https://www.swebok.org

[3] M. Dorodchi and N. Dehbozorgi, "Utilizing open source software in teaching practice-based software engineering courses," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2016, pp. 1–5.

[4] L. Deng, J. Dehlinger, and S. Chakraborty, "Teaching software testing with free and open source software," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Oct. 2020, pp. 412–418.

[5] A. N. Kumar, R. K. Raj, S. G. Aly, M. D. Anderson, B. A. Becker, R. L. Blumenthal, E. Eaton, S. L. Epstein, M. Goldweber, P. Jalote, D. Lea, M. Oudshoorn, M. Pias, S. Reiser, C. Servin, R. Simha, T. Winters, and Q. Xiang, *Computer Science Curricula 2023*. New York, NY, USA: Association for Computing Machinery, 2024.

[6] M. Arlitt, T. Coughlin, P. Faraboschi, E. Frachtenberg, P. Laplante, D. Milojicic, N. Patel, and R. Saracco, "Future of the workforce," *Computer*, vol. 56, no. 1, pp. 52–63, Jan. 2023.

[7] A. Abedi, M. Amin, C. Amirat, J. Athavale, R. M. Badia, and M. Baker. (Jan. 2024). *Technology Predictions 2024*. IEEE Computer Society. Accessed: Feb. 23, 2025. [Online]. Available: https://ieeecs-media.computer.org/media/tech-news/tech-predictions-report-2024.pdf

[8] B. Dong, J. Bai, T. Xu, and Y. Zhou, "Large language models in education: A systematic review," in *Proc. 6th Int. Conf. Comput. Sci. Technol. Educ. (CSTE)*, Apr. 2024, pp. 131–134.

[9] OpenAI. (2023). *ChatGPT (GPT-3.5 Model)*. Accessed: Dec. 21, 2024. [Online]. Available: https://chat.openai.com/

[10] Microsoft. (2023). *Microsoft 365 Copilot*. Accessed: Dec. 21, 2024. [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/copilot

[11] Microsoft. (2023). *Microsoft 365 Copilot Overview*. Accessed: Dec. 21, 2024. [Online]. Available: https://learn.microsoft.com/en-us/copilot/microsoft-365/microsoft-365-copilot-overview

[12] G. Marvin, N. Hellen, D. Jjingo, and J. Nakatumba-Nabende, "Prompt engineering in large language models," in *Proc. Int. Conf. data Intell. Cognit. Informat.* Cham, Switzerland: Springer, Jan. 2024, pp. 387–402.

[13] L. Giray, ''Prompt engineering with ChatGPT: A guide for academic writers,'' *Ann. Biomed. Eng.*, vol. 51, no. 12, pp. 2629–2633, Dec. 2023, doi: 10.1007/s10439-023-03272-4.

[14] S. Haldar, M. Pierce, and L. F. Capretz, ''WIP: Assessing the effectiveness of ChatGPT in preparatory testing activities,'' in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Washington, DC, USA, Oct. 2024, pp. 1–5, doi: 10.1109/FIE61694.2024.10893214.

[15] M. Madan, M. Dave, and A. Tandon, ''Importance of RTM for testing a web-based project,'' in *Proc. 7th Int. Conf. Rel., INFOCOM Technol. Optim. (Trends Future Directions) (ICRITO)*, Aug. 2018, pp. 816–818.

[16] E. J. Uusitalo, M. Komssi, M. Kauppinen, and A. M. Davis, ''Linking requirements and testing in practice,'' in *Proc. 16th IEEE Int. Requirements Eng. Conf.*, Sep. 2008, pp. 265–270.

[17] C. Wang, F. Pastore, A. Goknil, and L. C. Briand, ''Automatic generation of acceptance test cases from use case specifications: An NLP-based approach,'' *IEEE Trans. Softw. Eng.*, vol. 48, no. 2, pp. 585–616, Feb. 2022.

[18] L. Naimi, E. M. Bouziane, M. Manaouch, and A. Jakimi, ''A new approach for automatic test case generation from use case diagram using LLMs and prompt engineering,'' in *Proc. Int. Conf. Circuit, Syst. Commun. (ICCSC)*, Jun. 2024, pp. 1–5.

[19] V. Bayrı and E. Demirel, ''AI-powered software testing: The impact of large language models on testing methodologies,'' in *Proc. 4th Int. Informat. Softw. Eng. Conf. (IISEC)*, Dec. 2023, pp. 1–4.

[20] R. Santos, I. Santos, C. Magalhaes, and R. de Souza Santos, ''Are we testing or being tested? Exploring the practical applications of large language models in software testing,'' in *Proc. IEEE Conf. Softw. Test., Verification Validation (ICST)*, May 2024, pp. 353–360.

[21] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, ''An empirical evaluation of using large language models for automated unit test generation,'' *IEEE Trans. Softw. Eng.*, vol. 50, no. 1, pp. 85–105, Jan. 2024.

[22] Y. Tang, Z. Liu, Z. Zhou, and X. Luo, ''ChatGPT vs SBST: A comparative assessment of unit test suite generation,'' *IEEE Trans. Softw. Eng.*, vol. 50, no. 6, pp. 1340–1359, Jun. 2024.

[23] G. Ryan, S. Jain, M. Shang, S. Wang, X. Ma, M. K. Ramanathan, and B. Ray, ''Code-aware prompting: A study of coverage-guided test generation in regression setting using LLM,'' *Proc. ACM Softw. Eng.*, vol. 1, pp. 951–971, Jul. 2024, doi: 10.1145/3643769.

[24] D. S. Rusdianto, H. Fabroyir, and U. L. Yuhana, ''Innovative approaches to impact analysis of requirement changes using LLM in software projects,'' in *Proc. IEEE Int. Symp. Consum. Technol. (ISCT)*, Aug. 2024, pp. 604–610.

[25] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, ''Software testing with large language models: Survey, landscape, and vision,'' *IEEE Trans. Softw. Eng.*, vol. 50, no. 4, pp. 911–936, Apr. 2024.

[26] H. Aludhilu and E. Sutinen, ''Bridging the gap: Addressing software testing challenges in Namibian startups through a tailored training approach,'' in *Proc. 7th ACM/IEEE Int. Workshop Softw.-Itensive Bus.* New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 79–86, doi: 10.1145/3643690.3648244.

[27] V. Garousi, A. Rainer, P. Lauvås, and A. Arcuri, ''Software-testing education: A systematic literature mapping,'' *J. Syst. Softw.*, vol. 165, Jul. 2020, Art. no. 110570.

[28] A. Siddique, G. M. Butt, A. Zahid, Q. N. Naveed, and M. T.-H. Alouane, ''Analyzing software industry trends to improve curriculum,'' *IEEE Access*, vol. 12, pp. 22510–22523, 2024.

[29] V. D. Kirova, C. S. Ku, J. R. Laracy, and T. J. Marlowe, ''Software engineering education must adapt and evolve for an LLM environment,'' in *Proc. 55th ACM Tech. Symp. Comput. Sci. Educ.* New York, NY, USA: Association for Computing Machinery, Mar. 2024, pp. 666–672, doi: 10.1145/3626252.3630927.

[30] Q. Lang, S. Tian, M. Wang, and J. Wang, ''Exploring the answering capability of large language models in addressing complex knowledge in entrepreneurship education,'' *IEEE Trans. Learn. Technol.*, vol. 17, pp. 2053–2062, 2024.

[31] Z. T. Hamad, N. Jamil, and A. N. Belkacem, ''ChatGPT's impact on education and healthcare: Insights, challenges, and ethical consideration,'' *IEEE Access*, vol. 12, pp. 114858–114877, 2024.

[32] J. Barambones, C. Moral, A. de Antonio, R. Imbert, L. Martínez-Normand, and E. Villalba-Mora, ''ChatGPT for learning HCI techniques: A case study on interviews for personas,'' *IEEE Trans. Learn. Technol.*, vol. 17, pp. 1460–1475, 2024.

[33] A. M. Abdelfattah, N. A. Ali, M. A. Elaziz, and H. H. Ammar, ''Roadmap for software engineering education using ChatGPT,'' in *Proc. Int. Conf. Artif. Intell. Sci. Appl. Ind. Soc. (CAISAIS)*, Sep. 2023, pp. 1–6.

[34] F. A. Pirzado, A. Ahmed, R. A. Mendoza-Urdiales, and H. Terashima-Marin, ''Navigating the pitfalls: Analyzing the behavior of LLMs as a coding assistant for computer science students—A systematic review of the literature,'' *IEEE Access*, vol. 12, pp. 112605–112625, 2024.

[35] R. F. Ramos, R. M. De Angel, A. P. Ruetas, A. C. Lagman, J. Morano, R. T. Payongayong, and J. C. Enrile, ''Effective lesson planning and assessment design using leveraging Microsoft copilot implementation,'' in *Proc. IEEE 15th Control Syst. Graduate Res. Colloq. (ICSGRC)*, Aug. 2024, pp. 331–336.

[36] F. Borović, K. Aleksić-Maslać, and P. Vranešić, ''Comparative analysis of generative AI tools in enhancing educational engagement,'' in *Proc. 47th MIPRO ICT Electron. Conv. (MIPRO)*, May 2024, pp. 514–519.

[37] T. Rahman, J. Nwokeji, R. Matovu, S. Frezza, H. Sugnanam, and A. Pisolkar, ''Analyzing competences in software testing: Combining thematic analysis with natural language processing (NLP),'' in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2021, pp. 1–9, doi: 10.1109/FIE49875.2021.9637220.

[38] F. Bilow and J. DeWaters, ''Using thematic analysis to characterize the connection between sociotechnical engineering courses and Students' sense of belonging and engineering identity,'' in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2023, pp. 1–9.

[39] A. Shoufan, ''Exploring students' perceptions of ChatGPT: Thematic analysis and follow-up survey,'' *IEEE Access*, vol. 11, pp. 38805–38818, 2023.

[40] C. Grimalt-Álvaro and M. Usart, ''Sentiment analysis for formative assessment in higher education: A systematic literature review,'' *J. Comput. Higher Educ.*, vol. 36, no. 3, pp. 647–682, Dec. 2024.

[41] D. K. Dake and E. Gyimah, ''Using sentiment analysis to evaluate qualitative students' responses,'' *Educ. Inf. Technol.*, vol. 28, no. 4, pp. 4629–4647, Apr. 2023.

[42] X. Liang, J. He, and Q. Sun, ''Implementation of automatic keyword extraction software for English text based on Python,'' in *Proc. 4th Int. Conf. Appl. Mach. Learn. (ICAML)*, Jul. 2022, pp. 129–132.

[43] A. Onan, S. Korukoğlu, and H. Bulut, ''Ensemble of keyword extraction methods and classifiers in text classification,'' *Expert Syst. Appl.*, vol. 57, pp. 232–247, Sep. 2016.

[44] D. Buenaño-Fernandez, M. González, D. Gil, and S. Luján-Mora, ''Text mining of open-ended questions in self-assessment of university teachers: An LDA topic modeling approach,'' *IEEE Access*, vol. 8, pp. 35318–35330, 2020.

[45] N. Grönberg, A. Knutas, T. Hynninen, and M. Hujala, ''Palaute: An online text mining tool for analyzing written Student course feedback,'' *IEEE Access*, vol. 9, pp. 134518–134529, 2021.

[46] E. P. Enoiu, ''Teaching software testing to industrial practitioners using distance and web-based learning,'' in *Proc. 1st Int. Workshop Frontiers Softw. Eng. Educ.*, Villebrumier, France. Berlin, Germany: Springer, Jan. 2020, pp. 73–87, doi: 10.1007/978-3-030-57663-9_6.

[47] Money Manager Ex Development Team. (2025). *Money Manager Ex: GitHub Repository*. Accessed: Jan. 23, 2025. [Online]. Available: https://github.com/moneymanagerex

[48] M. Kanagavel and M. Contributors. (2024). *Money Manager Ex (MMEX) User Manual*. Includes Copyright 2005–2009 by M. Kanagavel and 2013–2019 by MMEX Contributors. Accessed: Jan. 23, 2025. [Online]. Available: https://moneymanagerex.org/moneymanagerex/enUS/index.html

[49] C. Hutto and E. Gilbert, ''VADER: A parsimonious rule-based model for sentiment analysis of social media text,'' *Proc. Int. AAAI Conf. Web Social Media*, vol. 8, no. 1, pp. 216–225, May 2014.

[50] *Natural Language Toolkit (NLTK) Documentation, Online Documentation, NLTK Project, 2024, version 3.9.1*. Accessed: Dec. 31, 2024. [Online]. Available: https://www.nltk.org/

[51] B. Hamid and N. Ikram, ''Industry perceptions of the competencies needed by novice software tester,'' *Educ. Inf. Technol.*, vol. 29, no. 5, pp. 6107–6138, Apr. 2024, doi: 10.1007/s10639-023-12055-2.

SUSMITA HALDAR (Member, IEEE) received the bachelor's and master's degrees in computer science from Concordia University, Montreal, QC, Canada. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Western University, London, Canada.

From 2005 to 2006, she was a Research Assistant at the Department of Computer Science, Western University. From 2006 to 2019, she worked in the industry in the domains of software testing, application development, business systems analysis, and project management. Since 2019, she has been a Professor at the School of Information Technology, Fanshawe College, London, ON, Canada. She is currently the Program Coordinator and a Professor of the Post Graduate Certificate Program in Software and Information Systems Testing, Fanshawe College. Her research interests include software testing education, scholarship in teaching and learning, large language models, and applications of machine learning and artificial intelligence in the software testing domain.

MARY PIERCE received the B.A. degree (Hons.) in economics from the University of Toronto and the M.A. degree in education from Central Michigan University.

She has been the Dean of the Faculty of Business, Information Technology and Part-Time Studies, Fanshawe College, London, ON, USA, since 2000. Prior to her career in education, she had a senior-level career in business with Procter and Gamble, S. C. Johnson Wax, The Blackburn Group, and Citigroup International. She retired from the Fanshawe College, in December 2024. She has been innovative and tireless in her efforts to build a dynamic and thriving academic environment. She has been a Leader in the development of online education, international exchange programs for students, pathway opportunities, and community partnerships. She has been the Chair of the provincial ONCAT transfer project for business programs in Ontario and the past Chair of the Ontario Heads of Business. She was a key player in the development of the original transfer agreements with the Institutes of Technology, Ireland, now the Technological Universities of Ireland, and has led numerous, exchange and project initiatives with Irish Partners, and being a champion of the Irish credit transfer agreements.

Ms. Pierce was honored by OCASA as the Researcher of the Year for her work in Academic Integrity, in 2018. She was recognized with the National Leadership Excellence Award by Colleges and Institutes Canada, in 2015. She was honored as a Researcher of the Year by OCASA, in 2018, for her thesis research work in Academic Integrity.

LUIZ FERNANDO CAPRETZ (Senior Member, IEEE) has vast experience in the software engineering field as a Practitioner, the Manager, and an Educator. Before joining Western University, Canada, he worked at both technical and managerial levels, taught, and did research on the engineering of software in Brazil, Argentina, U.K., Japan, United Arab Emirates, Malaysia, and Singapore. He is currently a Full Professor of software engineering and the Director of the Software Engineering Program. He has held visiting positions at the University of Sharjah, New York University, Abu Dhabi, Universiti Teknologi PETRONAS, and Yale-NUS College. His research interests include software engineering, human aspects of software engineering, software testing, and generative AI for software engineering. For further information, please visit: http://eng.uwo.ca/electrical/faculty/capretz_l