Generating Exploratory Test Suites with ChatGPT: Insights from an Empirical Study

Natália Salvino André
Federal University of Campina Grande
Campina Grande, Brazil
natalia.andre@ccc.ufcg.edu.br

Everton L. G. Alves
Federal University of Campina Grande
Campina Grande, Brazil
everton@computacao.ufcg.edu.br

ABSTRACT

Exploratory Testing (ET) plays a critical role in ensuring software quality by uncovering edge cases and enabling adaptive evaluation of systems. However, ET is often constrained by its reliance on tester expertise, challenges in reproducing failures, and limited testing resources. Generative Artificial Intelligence (GenAI), particularly ChatGPT, presents opportunities to support ET by automating tasks such as test case generation. This paper reports on an empirical study comparing exploratory test suites generated by ChatGPT using six image-based prompts with those created by human testers. We evaluated the quality of the generated suites, identified the most effective prompt formats, and gathered feedback from professional testers. Results show that ChatGPT can produce high-quality test suites, though it may miss complex faults and occasionally generate incoherent cases. The best performance was achieved using sub-prompts focused on specific features. Testers found the AI-generated suites clear and practical but noted gaps requiring complementary manual scenarios. These findings suggest that ChatGPT can serve as a valuable aid in ET, while highlighting areas for further improvement.

KEYWORDS

Exploratory Testing, GenAI, ChatGPT, Manual Testing.

1 Introduction

Testing is a crucial aspect of software development, performed to enhance a product's value by assessing its quality and reliability. Testing activities involve identifying and addressing faults and improving the overall performance of a software [21]. The primary goals of testing are not only to verify the software's behavior but also to detect faults early in the development process, thereby reducing the cost and effort required for later fixes. Furthermore, testing mitigates risks related to security, performance, and system reliability, preventing critical issues such as data breaches and system failures [3].

Exploratory Testing (ET) [16] is an approach where testers actively explore the application, often through its GUI, using their knowledge and experience to discover defects and identify issues without predefined test scripts. ET emphasizes the tester's freedom, allowing the design, execution, and interpretation of the test results to occur concurrently without adhering to a predefined script. This approach is particularly valuable in scenarios with limited testing time or insufficient system documentation. It enables the discovery of extreme scenarios that might not have been previously identified, fostering a dynamic and adaptable approach to software evaluation. Despite the great focus on scripted automated testing in research, ET remains indispensable in the software industry [13, 22].

It is important to clarify that ET does not imply a lack of preparation [15]. Effective ET requires testers to plan, gather, comprehend, and analyze extensive information about application domains, software products, and user needs. To support ET, concrete principles, guidelines, and methodologies have been established [8, 15, 27]. For instance, Session-Based Test Management (SBTM) [17] structures ET sessions, while recent work [24] has applied rule-based heuristics to generate test scenarios by combining insights from relevant bug reports. However, such strategies often fail to gain practical adherence from testers due to challenges such as complexity, high costs, and the lack of proper tools.

ET also presents practical challenges [8, 22]. Its effectiveness relies heavily on the tester's experience and creativity. Moreover, the lack of structured test cases can make reproducing identified failures difficult, potentially compromising the quality of both the tests and the final software. In contexts with limited resources, testers may have to juggle multiple roles, such as development or technical support, resulting in a lack of specialization that can further reduce test effectiveness.

Given these challenges, generative artificial intelligence (GenAI) [12] has the potential to significantly assist ET. Large Language Models (LLMs) are AI systems trained on vast textual datasets to understand and generate human-like language [11]. These systems can use and generate content such as images, texts, and audio, while improving their outputs through examples [12]. In ET, LLMs may enhance productivity by automating tasks such as generating test cases, crafting realistic user scenarios, and identifying edge cases based on system requirements.

One of the most popular generative AIs is ChatGPT [20], known for its ability to handle diverse tasks effectively. It incorporates the latest advancements in natural language processing, making it well-suited for understanding complex queries and generating human-like responses [12]. Recent versions of ChatGPT (e.g., GPT-4o) support more complex inputs, such as images and videos. This new capability may open up promising opportunities for enhancing exploratory testing (ET) and automating various aspects of the testing process. Visual information provides richer context for understanding software behavior, potentially enabling LLMs to generate test cases that better reflect real-world scenarios. This combination leverages both the reasoning power of LLMs and the descriptive richness of images, potentially bridging gaps between manual and automated testing practices. However, the full potential of this recent technology — and how to best apply it in the context of ET - still needs to be fully assessed.

In this paper, we present an empirical study on the use of ChatGPT to generate exploratory test suites. We compare test suites created by testers with those generated by ChatGPT using six different prompts, which include descriptive images of the system under test. We assess the quality of the generated suites and identify which prompts yield the best results. Additionally, we conduct a survey with professional testers to gather their opinions on the ChatGPT-generated test suites.

A summary of our key findings:

- ChatGPT is capable of generating high-quality exploratory test suites, although with some limitations. Issues such as nonsensical tests (due to AI hallucinations) and a tendency to overlook more complex faults were observed.
- The highest-quality test suites were produced when using sub-prompts tailored to each feature.
- Most testers found the ChatGPT-generated suites easy to understand and effective. While they indicated they would include these suites in exploratory testing sessions, they also noted that some important scenarios were not adequately covered.

Our findings provide insights to help testers decide whether and how to leverage ChatGPT effectively for exploratory testing.

The remainder of this paper is structured as follows. Section 2 introduces key concepts relevant to the study. Section 3 describes the methodology used in the empirical study. Section 4 discusses the results. Section 5 addresses potential threats to the validity of the study. Section 6 presents related work. Finally, Section 7 concludes the paper.

2 Background

This section presents an overview of key concepts relevant to our study.

2.1 Exploratory testing

Exploratory testing (ET) is a software testing approach where learning, test design, and execution are performed simultaneously. Unlike more structured testing methods, it is less formal and often involves random exploration, making it effective at uncovering faults that might otherwise go unnoticed [9]. This method is particularly advantageous when there is a need to quickly gain insights into a product or application and provide immediate feedback. By evaluating the product's quality from the user's perspective, exploratory testing helps identify new test scenarios, thereby improving test coverage and enhancing overall software quality [9].

To support ET, various guidelines and methodologies have been proposed [8, 15, 27]. These approaches mostly focus on providing frameworks and heuristics that help testers structure and define ET test cases. However, such strategies have yet to gain widespread adoption, as ET remains largely an ad-hoc practice.

2.2 Large Language Models

Large Language Models (LLMs) are advanced artificial intelligence systems trained on massive textual corpora to perform a wide variety of language-based tasks [11]. By learning statistical patterns and semantic structures from their training data, LLMs are capable of interpreting, generating, and reasoning with human language in a way that often mirrors human communication. Their capabilities include understanding context, producing coherent and relevant responses, translating between languages, summarizing information, and more.

LLMs operate using billions (or even trillions) of parameters, which enable them to capture nuanced relationships between words, phrases, and concepts. This computational scale allows them to exhibit surprising levels of fluency, coherence, and contextual awareness, making them foundational technologies in modern natural language processing (NLP) [11]. Despite their versatility, LLMs may still exhibit limitations, such as hallucination (i.e., producing incorrect or fabricated information), sensitivity to prompt phrasing, and biases inherited from training data.

Popular LLMs include ChatGPT[1], known for its versatility in creative and coding tasks. Bard[2] is widely used for search optimization and provides seamless integration with Google's ecosystem. LLaMA models[3] are designed for research and academic needs, while Claude[4] emphasizes safety and interpretability. Smaller open-source models, such as those from Hugging Face[5] and EleutherAI[6], offer accessible and customizable options.

2.3 ChatGPT

ChatGPT is a state-of-the-art large language model designed to facilitate human-like interactions with users through a chatbot and/or API interface. Built upon the GPT (Generative Pre-trained Transformer) architecture, ChatGPT is trained on a vast and diverse corpus of internet text, allowing it to understand and generate contextually appropriate responses to user inputs [18]. The specificity and clarity of a user's prompt play a critical role in shaping the quality, focus, and usefulness of the generated content.

A key feature of ChatGPT is its user-friendly interface. Users can tailor interactions by specifying the desired tone (e.g., formal, casual), style (e.g., academic, explanatory), and output format (e.g., bullet points, code snippets). In addition, ChatGPT allows control over the level of detail, complexity, and language of the response, making it adaptable to a wide range of domains—from casual conversation to professional and technical applications[18]. The latest generation, such as GPT-4o, extends these capabilities further by supporting multimodal prompts, including combinations of text, images, audio, and video, thus broadening its applicability to visually rich or cross-modal tasks.

Despite its strengths, ChatGPT has limitations. It may produce responses that sound plausible but are incorrect or nonsensical. It is also sensitive to variations in phrasing, sometimes yielding inconsistent answers to similar prompts. Additionally, it tends to generate overly verbose or repetitive outputs, a behavior influenced by training biases favoring longer responses [7].

3 Empirical Study Design

We designed an empirical study to investigate the use of ChatGPT for generating exploratory test suites based on different prompts and to evaluate the quality of the generated suites.

[1]https://chatgpt.com/
[2]https://blog.google/technology/ai/try-bard/
[3]https://ai.meta.com/blog/large-language-model-llama-meta-ai/
[4]https://www.anthropic.com/news/claude-3-family
[5]https://huggingface.co/
[6]https://www.eleuther.ai/

We selected ChatGPT for this study due to its widespread adoption and its alignment with the core demands of exploratory testing, which relies heavily on human-like reasoning, adaptability, and contextual understanding—capabilities that ChatGPT effectively simulates through advanced natural language generation. Its ability to interpret diverse prompts, generate realistic test ideas, and adapt responses to varying scenarios makes it particularly well-suited for investigating how generative AI can support the flexible and unscripted nature of exploratory testing.

To guide the study, we formulated the following research questions:

RQ1 Can ChatGPT generate good exploratory test suites?
This question aims to evaluate the quality of the suites generated by ChatGPT through different metrics and compare them with manually written suites. Through this analysis, we seek to understand whether ChatGPT's suites are as effective as manual ones and if they can offer practical advantages to testers.

RQ2 Which prompt provides the best results for generating exploratory test suites?
The focus of this question is to explore different ways testers can use ChatGPT to generate exploratory tests, evaluate them, and identify which approach yields the best results. From this, we aim to extract and propose a format that testers can use in future testing activities.

RQ3 What are the testers' perspectives on the test suites generated by ChatGPT?
This question seeks to gather insights from testers regarding the quality and applicability of the test suites produced by ChatGPT. With their feedback, we aim to understand whether the proposed use of ChatGPT can assist them in generating tests and identify points that might discourage its usage.

3.1 Methodology

Our study was conducted in five steps (Figure 1). First, we defined six prompt strategies to be used for generating exploratory test suites. Next, we selected five target systems and used ChatGPT to generate one test suite for each combination of prompt and system. To enable comparison with human-created suites, we invited professional testers to participate in exploratory testing sessions, during which they manually designed and executed test cases over the course of one hour. These testers documented their test cases using structured report spreadsheets. We then executed and evaluated all ChatGPT-generated test suites, comparing them to the manual ones using a set of predefined metrics. Finally, we conducted a survey with professional testers to gather their perceptions of the quality, usefulness, and coverage of the ChatGPT-generated test suites.

3.1.1 Participants. A total of 13 testers participated in the study. Participants were recruited through social media platforms, which is a common and effective strategy for reaching professional software developers in empirical studies. To be eligible, participants had to meet two inclusion criteria: (i) they must be professionals currently working in testing-related tasks, and (ii) they must have had at least some prior exposure to exploratory testing. These criteria ensured that participants had sufficient background to understand the study tasks and provide relevant insights. Only individuals who met these requirements were invited to take part in the study.

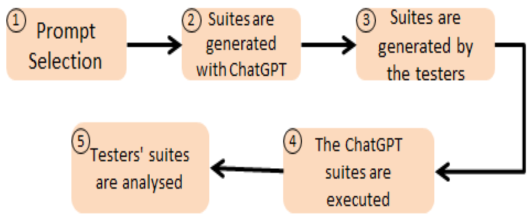

1 Prompt Selection
2 Suites are generated with ChatGPT
3 Suites are generated by the testers
4 The ChatGPT suites are executed
5 Testers' suites are analysed

Figure 1: Study steps.

Five participated in both the generation/execution of manual suites (step 3) and the survey (step 5), while the other eight participated only in the survey (step 5). The participants had varying levels of professional experience. Most work in software projects: 46.2% have between 3 and 6 years of professional experience, while 30.8% have between 1 and 3 years. Additionally, 61.5% stated they create and execute tests in their daily work routine, and 46.2% reported conducting exploratory tests frequently.

3.1.2 Test Case Format and Metrics. To ensure consistency, we defined a standardized test case format to be used throughout the study. Each test case includes the following fields: *description*, *prerequisites*, *steps for reproduction*, and *expected results*. The *description* field provides a concise summary of the objective the test aims to verify. The *prerequisites* field lists any conditions that must be met prior to execution, which are essential for ensuring test reproducibility. The *steps for reproduction* field outlines the specific actions required to perform the test, supporting clarity and repeatability. Finally, the *expected results* field describes the anticipated system behavior, serving as the reference for determining whether the test has passed or failed.

To help answer our RQs, we defined the following metrics: *total number of tests*, *average number of test case steps*, *number of nonsensical tests*, *number of identified faults*, and *test suite execution time*. The *total number of tests* represents the total count of test cases in the suite, providing an overview of its scope. The *average number of test case steps* measures the mean number of steps per test case, offering insight into the complexity of the tests. The *number of nonsensical tests* quantifies the hallucinations within the generated suites. The *number of identified faults* serves as an indicator of the test suite's effectiveness in uncovering system faults. Lastly, the *test suite execution time* captures the total time required to execute all test cases, reflecting the efficiency of the suite. All metrics were manually collected by the first author and were used to assess the overall quality of the test suites.

3.1.3 Systems, Faults, and Testing Instructions. As subjects for our study, we selected five systems: Readme.so [6], Par de Jarro [5], AnkiDroid [2], ActivityDiary [1], and Omni-notes [4]. Readme.so is an open-source project, while Par de Jarro is a proprietary project with previously mapped faults.

For Readme.so, due to its simplicity, we instructed both ChatGPT and the testers to evaluate the entire system. Consequently, the prompts covered all identified functionalities, enabling the testers to explore all features as well. In the case of Par de Jarro, the focus was on testing five pre-defined features: *registering a new user*, *logging in*, *logging out*, *viewing the user profile*, and *editing the user profile*.

AnkiDroid, ActivityDiary, and Omni-notes were selected from the Themis Benchmark [23], a repository of Android projects that provides system APKs with mapped faults, allowing for their reproduction. The benchmark also categorizes the difficulty of reproducing these faults as easy, medium, or difficult. In our study, the AnkiDroid APK contained a fault with a *difficult* reproduction level, ActivityDiary had a *medium-level* fault, and Omni-notes included an *easy-level* fault. For these systems, we focused the generated tests (both ChatGPT and manual) on the specific features associated with the identified faults. It is important to highlight that we randomly associated the testers to which system they should test.

3.1.4 ChatGPT and Prompt Selection. To assist in selecting the prompts for our study, we created a script[7] that interacts with the OpenAI application programming interface (API)[8] and records the responses. For this purpose, we decided to use the GPT-4o model, which supports a combination of text, audio, images, and video as input.

A total of six prompts were selected, all based on the Chain of Thoughts strategy [26], but varying in the level of information provided and complexity. Each prompt consisted of a textual message and a set of images encoded in Base64[9]. The textual messages followed a consistent structure: they instructed ChatGPT to assume the role of a tester, indicated that system images were included in the input, and specified the format for the test cases. We did not impose any constraints on the size of the generated test suites. The first prompt served as the foundation, with the others being developed incrementally. Details of each prompt are provided below, and Table 1 presents examples of how each prompt was used.

- **Prompt 1**: The simplest prompt. It does not specify the functionalities to be tested. Consequently, the accompanying images are more general, displaying various screens of the system.
- **Prompt 2**: This prompt includes guidance on what aspects should be covered in the tests. However, since it still does not specify particular functionalities, the provided images also show multiple screenshots of the system.
- **Prompt 3**: Similar to Prompt 2, this prompt offers general guidance, but introduces images edited to include tags describing details of the system. An example of this tagging approach is shown in Figure 2.
- **Prompt 4**: This and subsequent prompts utilize the sub-prompt strategy, where a distinct and separate prompt is generated for each feature to be tested.
- **Prompt 5**: Building on the structure of Prompt 4, this prompt adds a new segment to the sub-prompts, including a brief description of the application to be tested.
- **Prompt 6**: Following the same pattern as Prompt 5, this prompt introduces an additional segment in the sub-prompts, describing the test environment to be considered. This addition aims to avoid test cases referencing tools unavailable to the user.

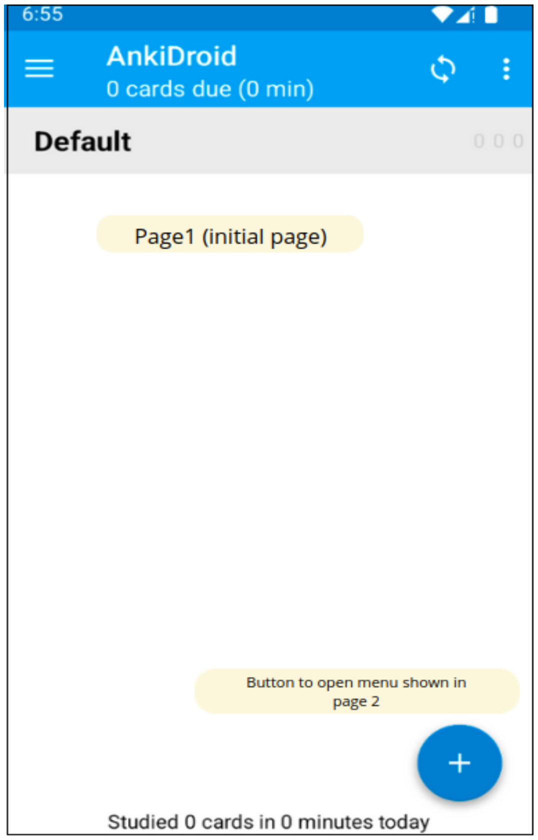

6:55
AnkiDroid
0 cards due (0 min)
Default
Page1 (initial page)
Button to open menu shown in page 2
Studied 0 cards in 0 minutes today

Figure 2: Tagged screenshot of the AnkiDroid sent in Prompt 3

3.1.5 Survey. To understand testers' perspectives on exploratory test suites generated by ChatGPT, we conducted a survey in which we presented test suites generated for individual features of our target systems and asked testers questions regarding the quality and applicability of each suite. The test suites used in the survey were as follows: for *readme.so*, the suite for the *add custom* feature; for *Par de jarro*, the *login* suite; for *Omni-notes*, the *create note* suite; for *ActivityDiary*, the *edit location service* suite; and for *AnkiDroid*, the *create card* suite.

The test suites referenced were generated using the prompt that yielded the best results in our study. It is important to highlight that we did not disclose to the participants whether the presented suites were manually or automatically generated. The survey included six Likert scale [14] questions and two open-ended questions:

(1) The suite is easy to understand.
(Totaly Agree – Agree – Neither agree or disagree – Disagree – Totaly disagree)
(2) The suite is effective for testing the functionality.
(Totaly Agree – Agree – Neither agree or disagree – Disagree – Totaly disagree)

[7]https://github.com/natalia-sa/Exploratory-testing-with-ChatGPT
[8]OpenAI API reference: https://platform.openai.com/docs/api-reference/introduction
[9]Base64 encoding: https://www.freecodecamp.org/news/what-is-base64-encoding/

Table 1: Examples of prompts used to generate exploratory test cases.

Prompt	**Text**
Prompt 1	From the point of view of a tester, generate exploratory test cases for the AnkiDroid app. I will provide screenshots of the app GUI. The test cases should follow the structure below: • test number • Description: The test case description • Prerequisites: Specifies the conditions that must be met before executing the test steps • Steps: Enumerated steps to execute the test • Expected results: The expected test results
Prompt 2	From the point of view of a tester, give me exploratory test cases for the AnkiDroid app. Consider unusual flows, try to find possible bugs, failures, etc. I will provide screenshots of the app GUI. The test cases should follow the structure below: • test number • Description • Prerequisites • Steps • Expected results
Prompt 3	From the point of view of a tester, give me exploratory test cases for the AnkiDroid app. Consider unusual flows, try to find possible bugs, failures, etc. I will provide screenshots of the app GUI. The images contain tags that identify some features and characteristics of the app. The test cases should follow the same structure: • test number • Description • Prerequisites • Steps • Expected results
Prompt 4	From the point of view of a tester, give me exploratory test cases for the "add custom section" feature in the Readme.so app. Consider unusual flows, try to find possible bugs, failures, etc. I will provide screenshots of the app GUI. The test cases should follow the defined structure: • test number • Description • Prerequisites • Steps • Expected results
Prompt 5	From the point of view of a tester, give me exploratory test cases for the "add custom section" feature in the Readme.so app. This app creates README files in Markdown. It does not use a database and does not require user login. Consider unusual flows, bugs, failures, and security issues. I will provide screenshots of the app GUI. Follow the given structure: • test number • Description • Prerequisites • Steps • Expected results
Prompt 6	From the point of view of a tester, give me exploratory test cases for the "add custom section" feature in the Readme.so app. Assume the tests will be performed on Linux using Firefox and Chrome only. The app does not use a database or user authentication. Look for unusual flows, bugs, failures, or security issues. Follow the defined structure: • test number • Description • Prerequisites • Steps • Expected results

(3) The test suite satisfactorily covers the functionality.
(Totaly Agree – Agree – Neither agree or disagree – Disagree – Totaly disagree)

(4) I would include the test suite in an exploratory testing session.
(Totaly Agree – Agree – Neither agree or disagree – Disagree – Totaly disagree)

(5) There is no need to add more test cases to the suite.
(Totaly Agree – Agree – Neither agree or disagree – Disagree – Totaly disagree)

(6) There is no need to remove test cases from the suite.

(Totaly Agree – Agree – Neither agree or disagree – Disagree – Totaly disagree)

(7) What positive aspects did you identify in the test suite?
(8) What negative aspects did you identify in the test suite?

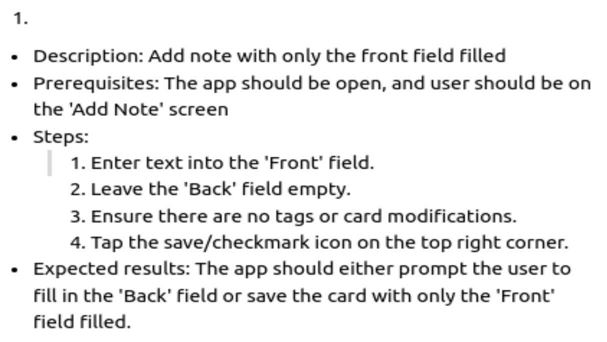
4 Results and Discussion

The results collected in the different stages of this study were used to answer each of the research questions and are presented below. The artifacts from this study (including projects, generated test suites, scripts, etc.) are available in our repository[10].

4.1 Can ChatGPT generate good exploratory test suites?

Table 2 presents the results collected for each metric and system. Regarding the total number of tests, we observed that for *Omni-notes*, *ActivityDiary*, and *AnkiDroid* – where we requested tests that explored a single feature – we found the highest number of tests in the manual suites. For the *Readme.so* and *Par de Jarro* systems – where more than one feature was tested – prompts 4, 5, and 6, which consisted of sub-prompts, generated a significantly higher number of tests than the manual suites.

Regarding the average number of test case steps, the lowest average generated by ChatGPT was 1.36 for *Readme.so*, and the highest was 7.10 for *AnkiDroid*. The averages generated by manual tests were surpassed by ChatGPT in 2 out of the 5 systems: *Readme.so* and *AnkiDroid*. In general, the steps in the prompt suites clearly indicated what needed to be done to reproduce the tests. An example of a test case generated using Prompt 4 for the *add note* feature in *AnkiDroid* is shown in Figure 3.



1.

- Description: Add note with only the front field filled
- Prerequisites: The app should be open, and user should be on the 'Add Note' screen
- Steps:
  1. Enter text into the 'Front' field.
  2. Leave the 'Back' field empty.
  3. Ensure there are no tags or card modifications.
  4. Tap the save/checkmark icon on the top right corner.
- Expected results: The app should either prompt the user to fill in the 'Back' field or save the card with only the 'Front' field filled.

Figure 3: Test case generated to AnkiDroid

We called nonsensical tests the ones that could not be executed, as they include steps/actions that were impossible to replicate. This can occur due to hallucinations commonly exhibited by generative AIs. An example of a nonsensical test case generated for *Par de Jarro* is shown in Figure 4. We classified it as nonsensical because there is no "Save" button on the edit profile page. The highest percentage of nonsensical tests was generated by Prompt 1 for *AnkiDroid*, where 100% of the tests were nonsensical because they referred to elements that were not existing in the system (e.g., buttons that were not part of the GUI). The second highest percentage of nonsensical tests was 33.33%, also generated by Prompt 1, but for *Par de Jarro*. However, the highest number of nonsensical tests was 29, generated by Prompt 4 for *Readme.so*, representing 24.17% of the test suite.

[10]https://github.com/natalia-sa/Exploratory-testing-with-ChatGPT/tree/main

Regarding the total number of identified faults, the first author manually inspected all suite executions. In 4 out of the 5 systems, the highest number of faults was found by the manual sessions. Only Prompt 5 for the *Omni-notes* system identified the same number of faults as the manual sessions. However, in 3 out of the 5 systems, the test suites generated from a single prompt were able to identify more faults than at least one of the manual sessions. For the projects taken from the Themis Benchmark, which contained faults of varying reproduction difficulty, the generated tests identified only the easy-to-reproduce ones. Nevertheless, in some cases, the generated tests identified faults that were not reported in the manual sessions, suggesting that ChatGPT could be used to complement manual test suites.

Finally, regarding the test suite execution time, the largest time differences between the ChatGPT-generated test suites and the manual suites occurred for *Readme.so* and *Par de Jarro* using Prompts 4, 5, and 6. This was because more than one feature was tested, and a separate prompt was sent for each feature, resulting in a large volume of tests. Even in these cases, the execution times remained close to the one-hour window allocated for testing sessions. Thus, in general, the time spent on the prompt-generated suites did not pose a barrier to their potential practical use by testers.

The data collected suggests that ChatGPT is capable of generating effective exploratory test suites, though with some limitations. The prompts that generate one suite for each feature performed well in generating a significant number of test cases with well-defined steps, and the execution times were close to the manual session limits. However, issues such as nonsensical tests, caused by AI hallucinations, and a tendency to miss faults with higher reproduction difficulty were noted. Although manual sessions generally identified more faults, ChatGPT suites discovered several that were missed during manual testing, which supports its potential as a complementary tool for exploratory testing. The readme.so app provides a clear example: only the test suites generated by prompts 5 and 6 showed that leaving a custom section title blank duplicates the previous section. Consequently, our findings suggest that ChatGPT is a promising tool for generating effective test suites that can augment, but not replace, manual testing efforts.

4.2 Which prompt provides the best results for generating exploratory test suites?

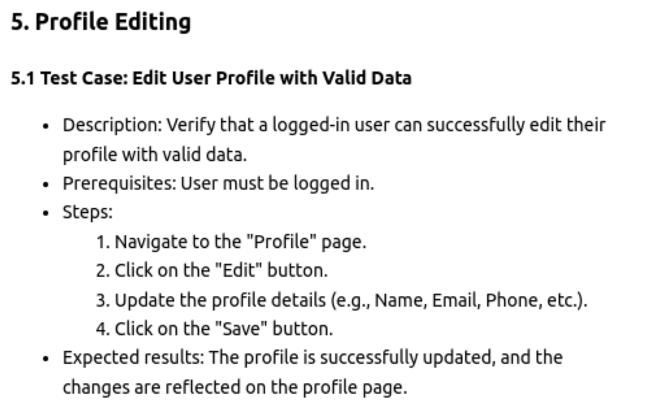
To answer this question, we compared the suites generated by the different prompts. As shown in Table 2, the prompts that performed best were those that generated a suite for each required feature. The prompt with the best overall performance was Prompt 5. It generated the highest number of tests for two of the five systems, as did Prompt 6, but produced a lower average number of test case steps for three of the five systems and four of its test suites were faster to execute than the ones generated by Prompt 6, indicating a balance between complexity and simplicity. It did not generate any nonsensical tests for three systems and showed low percentages for the other two: 4.03% for *Readme.so*, 6.25% for *Par de Jarro*. In terms of fault identification, it performed best for one system and tied for

Table 2: Test suite metrics

Test suite	Number of tests	Average number of test case steps	Number of non-sensical tests	Number of identified faults	Execution time (min)
Readme.so					
Manual tests 1	30	2.57	0	10	12.33
Manual tests 2	9	1.44	0	5	3.17
Prompt 1	11	1.36	0	2	3.50
Prompt 2	10	3	0	1	6.38
Prompt 3	12	3.08	0	3	6.95
Prompt 4	120	3.38	29	6	63
Prompt 5	124	3.15	5	7	65
Prompt 6	120	3.38	3	7	70
Par de Jarro					
Manual tests 1	25	4.08	0	10	11.75
Manual tests 2	15	5.80	0	5	6.33
Prompt 1	9	2.78	3	2	3.67
Prompt 2	10	2.90	1	3	4.48
Prompt 3	10	2.80	3	2	3.78
Prompt 4	42	4.21	11	6	28.41
Prompt 5	48	3.00	3	5	30.00
Prompt 6	51	3.74	3	6	38.75
Omni-notes					
Manual tests 1	14	4.14	0	2	4.30
Manual tests 2	21	7.33	0	5	10.72
Prompt 1	7	5.71	0	4	1.83
Prompt 2	8	4.38	0	4	2.17
Prompt 3	7	3.57	0	2	1.67
Prompt 4	6	4.83	0	4	1.67
Prompt 5	12	4.17	0	5	3.00
Prompt 6	11	5.45	0	3	3.15
ActivityDiary					
Manual tests 1	13	4.85	0	5	5.53
Manual tests 2	34	1.21	0	8	6.33
Prompt 1	5	5	0	0	2.13
Prompt 2	5	3	1	2	2.50
Prompt 3	5	4.60	1	2	2.17
Prompt 4	9	3.33	1	0	3.17
Prompt 5	9	4.11	0	2	2.80
Prompt 6	10	3.00	0	2	3.00
AnkiDroid					
Manual tests 1	14	4.14	0	1	6.47
Manual tests 2	10	2.00	0	1	4.25
Prompt 1	6	2.67	6	0	1.61
Prompt 2	8	3.88	2	0	2.67
Prompt 3	5	4.40	0	0	1.40
Prompt 4	8	3.12	1	0	4.41
Prompt 5	10	7.10	0	0	5.10
Prompt 6	10	5.50	1	0	4.50

the best result in three others with Prompt 6, being surpassed by Prompt 6 in one system.

Another key factor in prompt performance was the selection of images. Initial tests revealed that sending images of various parts of the system while the textual message focused on a single feature led to more hallucinations. Images that focused solely on one component of the screen, such as a modal or form, also caused similar issues. Therefore, for prompts 4, 5, and 6, where the features were specified, only images relevant to the feature were provided, and they encompassed the entire system screen. An example of an image used, which yielded better results, is shown in Figure 5.



5. Profile Editing

5.1 Test Case: Edit User Profile with Valid Data

- Description: Verify that a logged-in user can successfully edit their profile with valid data.
- Prerequisites: User must be logged in.
- Steps:
  1. Navigate to the "Profile" page.
  2. Click on the "Edit" button.
  3. Update the profile details (e.g., Name, Email, Phone, etc.).
  4. Click on the "Save" button.
- Expected results: The profile is successfully updated, and the changes are reflected on the profile page.

Figure 4: Nonsensical test case

Additionally, the time taken to create and submit the prompts to ChatGPT was brief, typically completed in just a few minutes. The textual prompts did not need to be complex to convey the context of the applications, as the images provided significant information. Moreover, gathering the images for Prompt 5 was also quick, requiring only system screenshots.

Thus, Prompt 5 demonstrated the best results for generating ET in our study. It achieved superior outcomes by using a sub-prompt for each feature to be tested, similar to prompts 4 and 6. However, the inclusion of a textual section that provided additional context about the application, which might not be explicit in the images, contributed to its improved performance. The images used were easy to obtain, required no editing, and the time spent creating each sub-prompt was minimal – no more than a few minutes.

4.3 What are the testers' perspectives on the test suites generated by ChatGPT?

This section presents the survey results on how testers evaluated the suites generated by ChatGPT using Prompt 5 (the best option based on our findings). Figure 6 displays the results for the six Likert scale questions.

When asked about the quality of the received test suite, the majority of testers found it easy to understand (100%), effective, and satisfactory for testing the feature (100%). Additionally, 92.3% agreed they would include the suite in a future exploratory testing session. Opinions on the completeness of the suite were more varied. While only 38% felt there was no need to add new test cases, 84.6% considered all provided test cases valuable, with no need to remove any. These results highlight the overwhelmingly positive feedback from testers regarding the ChatGPT-generated suites.

A deeper understanding of the testers' perspectives can be gained by analyzing their answers to the open-ended questions about the positive and negative aspects of the suites. Many testers highlighted the ease of understanding and the inclusion of edge cases. One tester stated:

> *The suite is sufficiently descriptive without being verbose. It is easy to understand what each test aims to verify. The steps are easy to reproduce.*

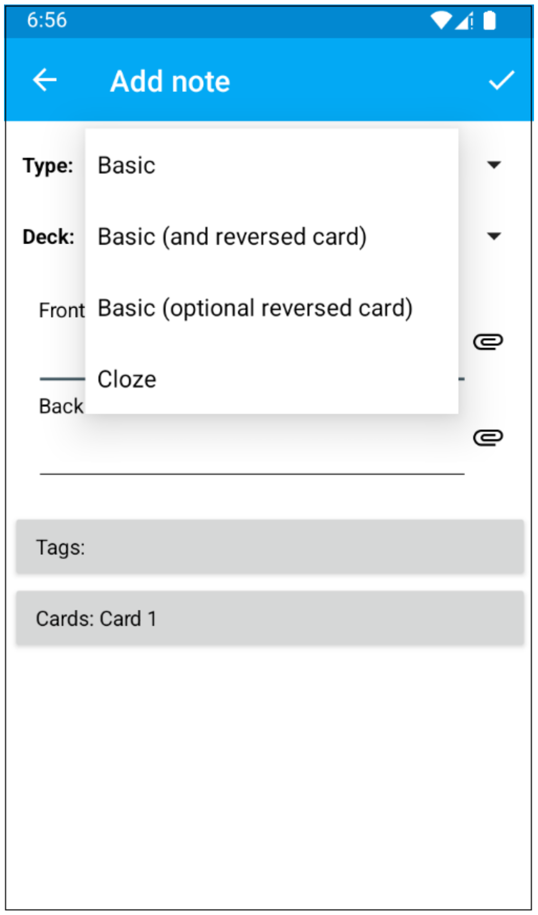


Figure 5: Screenshot of the AnkiDroid application sent as input to ChatGPT.

This type of feedback suggests that ChatGPT can produce well-structured and readable test cases, which are crucial in high-pressure testing contexts where time and clarity are paramount. The ability to generate human-understandable and reproducible steps demonstrates that LLM-generated test suites can lower the barrier to entry for testers with varying levels of expertise and potentially reduce the time required to design initial test plans.

Regarding the negative aspects, most testers pointed out tests that should have been included in the observed suites. One tester evaluating the suite for *AnkiDroid* emphasized:

> *I notice there is no security related tests, such as protection against malicious data input (e.g., script injection). Additionally, there are no tests aimed at evaluating the application's performance when handling large amounts of data, such as when creating many notes or displaying multiple items in the notes list.*

Such comment reflects a key limitation of LLM-generated test suites: they tend to focus primarily on surface-level, functional behaviors and may overlook deeper, system-level concerns such as security, performance, and stress scenarios. This reinforces the need for human oversight and complementary expertise when

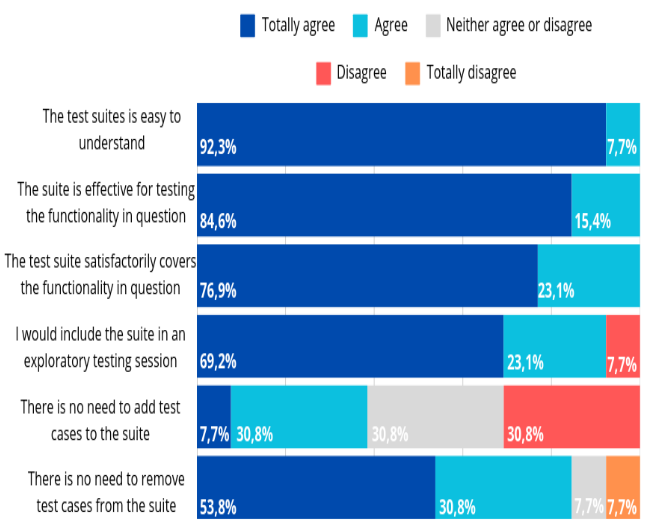


Figure 6: Testers' opinions on the suites generated by ChatGPT

using LLMs for exploratory testing. While ChatGPT may support the generation of a strong initial foundation for test design, it does not yet replace the domain knowledge and critical thinking that experienced testers bring, particularly when it comes to identifying and prioritizing non-obvious or context-specific risks.

These findings highlight the importance of hybrid approaches, combining the efficiency of LLM-generated content with human validation and enrichment, to ensure comprehensive and context-aware exploratory testing.

5 Threats to Validity

There are some threats to the validity of this study that we need to recognize. First, the data collected from the experiment is restricted to the specific systems used for testing: *Omni-notes*, *ActivityDiary*, *AnkiDroid*, *Readme.so*, and *Par de Jarro*, and the generalization of the findings to other systems is limited by this selection. However, these systems were chosen to encompass different types of features, covering faults with different levels of difficulty of reproduction. Despite this limitation, the study demonstrates a realistic scenario, and further research could extend these conclusions by including a larger set of systems.

The study involved a total of thirteen testers and only two manual testing sessions were executed for each system. Also, most of the participants work in software development. A larger number of participants, including more people working in the quality assurance field, could provide important comparisons for the study, but would make the study impractical. While this sample size may seem modest, it includes testers with varying levels of experience, and a significant percentage of them frequently perform exploratory testing, allowing for a reasonable representation of different perspectives.

Finally, this study relied on a specific LLM (ChatGPT) and a single version that was available at the time of testing (GPT-4o). As with all AI models, future versions may show improvements or differences in performance, which could alter the results. Nevertheless, the version used in this study was one of the most advanced and, at the time, made it relevant for the research, and the findings presented here are valid for understanding its current capabilities in generating exploratory test suites.

6 Related Work

In this section, we review previous studies that have explored similar topics related to the use of AI in software testing.

Guilherme and Vincenzi [10] conducted a study focused on evaluating the capability of ChatGPT to generate unit tests for Java programs, using metrics such as code coverage and mutation score. They automated the generation of test sets for 33 Java programs without human intervention, comparing the performance of ChatGPT model gpt-3.5-turbo with traditional automated tools. Their results showed comparable efficiency and effectiveness to existing tools. While this study investigates the use of ChatGPT for generating unit tests in Java, our research takes a different approach by focusing on exploratory testing in multiple systems, using ChatGPT model gpt-4o to generate test suites based on prompts with textual information and GUI images. This approach is relevant as it allows for a more dynamic and visual-based method of generating test scenarios, expanding the potential applications of LLMs in software testing.

Liu et al. [19] proposed GPTDroid, an approach for automated GUI testing of mobile apps that uses GPT-3 to generate human-like interactions. Their method formulates mobile GUI testing as a question-and-answer task, where GUI page information is passed to the LLM to elicit testing scripts. They evaluated GPTDroid on 86 apps from Google Play, demonstrating 71% activity coverage, 32% higher than the best baseline—and the ability to detect 36% more faults. Furthermore, they identified 48 new faults, with 25 of them confirmed or fixed. While this work focuses on improving testing coverage and fault detection using LLMs, our study seeks to relate ChatGPT with testers, comparing its performance with manual suites, investigating the best format for using the tool in the context of manual testing, and gathering testers' opinions on the generated material.

Zimmermann et al. [28] propose an approach for automating GUI testing in web applications by integrating GPT-4 with the Selenium framework, achieving significant improvements in branch coverage without the need for human-recorded data. Their study contrasts this approach with monkey testing, a method that generates random user inputs, and demonstrates that GPT-4 significantly outperforms it in terms of testing accuracy. While their work focuses on comparing GPT-4 with a baseline of random input generation, our study takes a different approach by comparing AI-generated test suites with manual ones. Instead of input techniques, we emphasize human feedback, gathering insights from testers who evaluate the test suites generated by ChatGPT using prompts that incorporate both textual and visual information.

Su et al. [25] investigates the use of Large Language Models (LLMs) to build a System Knowledge Graph (KG) of User Tasks and Failures. By combining system and bug knowledge from the KG with the logical reasoning abilities of LLMs, the approach generates exploratory test scenarios. This work diverges from our study and traditional exploratory testing, by requiring prior bug history to guide the LLM's process.

7 Concluding Remarks

Artificial intelligence has gained significant popularity in recent years and is now widely applied across various sectors. In the software industry, professionals harness its potential to streamline tasks and address common challenges. While recent research has explored AI, including ChatGPT, in software testing, its use for generating tests based on visual information remains underexplored.

This study investigated ChatGPT's potential to support ET. We evaluated the quality of generated suites, identified the most effective prompt format, and gathered testers' opinions. These insights are critical to understanding the tool's practical value and its integration into software testing activities.

Our results showed that ChatGPT-generated suites successfully identified faults across systems. Although manual suites found more faults overall, ChatGPT uncovered unique issues not detected manually. In 3 out of 5 systems, at least one ChatGPT prompt-generated suite identified more faults than a manual session. This suggests that the tool may be particularly useful when testers have limited experience or system knowledge—a common challenge in ET. Prompts that provided more context produced better results, helping ChatGPT understand scenarios and generate more relevant tests. Structuring prompts by feature and selecting full-screen, relevant images reduced hallucinations without making the suites overly long.

Testers praised the clarity and readability of the ChatGPT-generated suites—often a weakness in manual testing due to inconsistent documentation. Most participants said they would use these suites in manual sessions and did not find tests that should be removed. However, a common criticism was the absence of important test cases, with 30.8% of participants disagreeing that no additions were needed. This reinforces ChatGPT's role as a complementary tool that helps reveal gaps in manual testing.

Future work could expand the participant pool and evaluate other LLMs (e.g., Gemini, LLaMA, Mistral). Testing alternative prompt formats—such as video frame inputs or richer textual descriptions—may also improve results. Finally, exploring the trade-offs between prompt complexity and output quality can help optimize AI integration into testing workflows.

REFERENCES

[1] [n. d.]. ActivityDiary. https://github.com/ramack/ActivityDiary. Accessed: 2024-09-12.
[2] [n. d.]. AnkiDroid. https://github.com/ankidroid/Anki-Android. Accessed: 2024-09-12.
[3] [n. d.]. The Importance of Software Testing. https://www.computer.org/resources/importance-of-software-testing. Accessed: 2024-09-24.
[4] [n. d.]. Omni-Notes. https://github.com/federicoiosue/Omni-Notes. Accessed: 2024-09-12.
[5] [n. d.]. Par de Jarro. https://github.com/Par-de-Jarro. Accessed: 2024-09-12.
[6] [n. d.]. Readme.so. https://readme.so/en. Accessed: 2024-09-12.
[7] 2022. Introducing ChatGPT. https://openai.com/index/chatgpt/. Accessed: 2024-09-22.
[8] JD Cem Kaner and James Bach. 2006. The nature of exploratory testing. (2006).
[9] Deepak Parmar. [n. d.]. Exploratory testing. https://www.atlassian.com/continuous-delivery/software-testing/exploratory-testing. Accessed: 2024-09-21.
[10] Vitor Guilherme and Auri Vincenzi. 2023. An initial investigation of ChatGPT unit test generation capability. In *Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing*.
[11] IBM. [n. d.]. What are large language models (LLMs)? https://www.ibm.com/topics/large-language-models. Accessed: 2024-09-21.
[12] IBM. [n. d.]. What is generative AI? https://research.ibm.com/blog/what-is-generative-AI. Accessed: 2024-09-25.
[13] Juha Itkonen and Mika V Mäntylä. 2014. Are test cases needed? Replicated comparison between exploratory and test-case-based software testing. *Empirical Software Engineering* 19 (2014), 303–342.
[14] Ankur Joshi, Saket Kale, Satish Chandel, and D Kumar Pal. 2015. Likert scale: Explored and explained. *British journal of applied science & technology* 7, 4 (2015), 396–403.
[15] Cem Kaner. 2008. A tutorial in exploratory testing. *QUEST* 6 (2008).
[16] C. Kaner, J. Falk, and H.Q. Nguyen. 1999. *Testing Computer Software*. Wiley. https://books.google.com.br/books?id=Q-hTDwAAQBAJ
[17] Michael Kelly. 2018. Session-Based Test Management. In *How to Reduce the Cost of Software Testing*. Auerbach Publications, 119–136.
[18] Kristie Wright. 2023. ChatGPT large language model: Everything you need to know. https://indatalabs.com/blog/chatgpt-large-language-model. Accessed: 2024-09-22.
[19] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2023. Chatting with gpt-3 for zero-shot human-like mobile automated gui testing. *arXiv preprint arXiv:2305.09434* (2023).
[20] Megan Cerullo. 2023. ChatGPT is growing faster than TikTok. https://www.cbsnews.com/news/chatgpt-chatbot-tiktok-ai-artificial-intelligence/. Accessed: 2024-09-24.
[21] G.J. Myers, C. Sandler, and T. Badgett. 2011. *The Art of Software Testing*. Wiley. https://books.google.com.br/books?id=GjyEFPkMCwcC
[22] Dietmar Pfahl, Huishi Yin, Mika V Mäntylä, and Jürgen Münch. 2014. How is exploratory testing used? a state-of-the-practice survey. In *Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement*. 1–10.
[23] Ting Su, Jue Wang, and Zhendong Su. 2021. Benchmarking Automated GUI Testing for Android against Real-World Bugs. In *Proceedings of 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 119–130. doi:10.1145/3468264.3468620
[24] Yanqi Su, Zheming Han, Zhenchang Xing, Xin Xia, Xiwei Xu, Liming Zhu, and Qinghua Lu. 2022. Constructing a system knowledge graph of user tasks and failures from bug reports to support soap opera testing. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.
[25] Yanqi Su, Dianshu Liao, Zhenchang Xing, Qing Huang, Mulong Xie, Qinghua Lu, and Xiwei Xu. 2024. Enhancing exploratory testing by large language model and knowledge graph. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.
[26] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.
[27] James A Whittaker. 2009. *Exploratory software testing: tips, tricks, tours, and techniques to guide test design*. Pearson Education.
[28] Daniel Zimmermann and Anne Koziolek. 2023. GUI-Based Software Testing: An Automated Approach Using GPT-4 and Selenium WebDriver. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. doi:10.1109/ASEW60602.2023.00028