Image2Test: Using ChatGPT to Build Manual Tests from Screenshots

Manoel Aranda III
Federal University of Alagoas
Maceió, Brazil
mpat@ic.ufal.br

Antônio Wagner
Federal University of Alagoas
Maceió, Brazil
amcw@ic.ufal.br

Eduardo Barros
Federal University of Alagoas
Maceió, Brazil
eab@ic.ufal.br

Márcio Ribeiro
Federal University of Alagoas
Maceió, Brazil
marcio@ic.ufal.br

Alessandro Garcia
Pontifical Catholic University of Rio
Rio de Janeiro, Brazil
afgarcia@inf.puc-rio.br

Fabio Palomba
University of Salerno
Salerno, Italy
fpalomba@unisa.it

Baldoino Fonseca
Federal University of Alagoas
Maceió, Brazil
baldoino@ic.ufal.br

Ivan Machado
Federal University of Bahia
Salvador, Brazil
ivan.machado@ufba.br

ABSTRACT

Background: Website layouts often change with new design trends and front-end frameworks. Quality assurance is necessary during these changes, but manual testing takes much time and money. Manual tests are the standard way to maintain quality, but they are slow and expensive. Changes in graphical interfaces can cause errors or break features, which affects quality. Writing manual tests is the most time-consuming part of the process. **Aims**: This paper presents a tool that uses ChatGPT to create Natural Language Tests from screenshots and operator instructions. The goal is to reduce the time spent on manual test creation and to maintain quality in both the tests and the application. **Method**: We used two evaluation methods. First, we conducted a survey with 18 software testing professionals and students to compare tests made by ChatGPT and by humans. Second, we used Natural Language Processing techniques to measure the similarity between ChatGPT-generated tests and human-made tests. **Results**: The qualitative analysis showed that ChatGPT tests exceed human tests in completeness by a difference of 5.56%, achieving 36.11% acceptance rate. Human tests exceeded ChatGPT tests in clarity by 6.95%, reaching 41.67% acceptance rate. The quantitative analysis found that 66.7% of ChatGPT tests shared over 50% similarity with human tests. **Conclusions**: Our tool can help automate the creation of software tests. The similarity between AI-generated and human-made tests shows that this approach can save time and reduce costs, while keeping test quality at an acceptable level. This framework can help maintain quality during changes in website layouts and application development.

KEYWORDS

Natural Language Test, Manual Test, Test Case Generation, Software Testing, LLMs, ChatGPT

1 Introduction

In every aspect of software, quality should be a requirement. Because of this, most industrial software development requires at least some degree of testing. Whether in automated or manual setup, tests demand time to write and polish. According to the World Quality Report for 2023-2024 [9], more than 60% of all tests are not automated, i.e., are manual tests, in more than 80% of the enterprises worldwide. According to Fischbach *et al.*, manual tests require between 40 and 70 percent of the entire effort [21]. In terms of time-consuming activities, manual testing was ranked first in Perfecto's report [5], with 54% of the surveyed selecting it. So, we can safely say that time is a really valuable resource for the Quality and Assurance (Q&A) team, specially if most of the tests are manual, where the step-by-step describes ideally the complete interaction between the user and the system.

Preconditions: The user needs to be logged-in

#	Actions	Verifications
...	...	...
3	Click the Mozilla Firefox icon	Verify that the Mozilla Firefox browser opens.
...	...	...

Test A

Figure 1: Manual Test Example

An example of a natural language test is shown in Figure 1. In this test case, the user has to follow specific commands on a sequential matter, while also verifying the outputs of each command and satisfying the prerequisites. In an environment where tests are not priority, the Q&A team must test the software when it is in a ready to deploy state. Without access to code and in urgent need to test the application, the test engineer may not always guarantee the best test case possible.

According to reports [7, 9], enterprise workers perceive writing down manual tests as the most time-consuming part of the job, while also indicating a "lack of time to ensure quality". The time-efficiency problem related to writing automated tests can be solved by employing techniques that involve automatization (i.e., transforming manual processes in automatic) of all the software testing life-cycle [16, 18, 22, 44]. However, most of the solutions related to manual tests try to substitute them for automated tests altogether, whereas there are times when the manual test is preferred. In this context, there is a lack of studies that aim to help the Q&A worker in optimizing his time.

This paper aims to fill this gap, providing a tool to support the automatic creation of Manual Tests, using Large Language Models (LLMs), and then an evaluation of the proposed framework. The tool works by receiving two inputs:

- The objective of the test
- The screenshot of the system that will be tested

Annexing these inputs to a prompt that mimics the behavior of a manual tester while creating a natural language test, we can leverage a natural language test from a generative pre-trained transformer (GPT). In this paper, a specific GPT was selected, which was ChatGPT. After obtaining 12 tests from a sample of websites and specific objectives, we evaluate the produced tests by comparing them to human-made tests of the same websites and objectives.

To evaluate both the tool and the tests produced by it, this paper employs a two-fold empirical study. In Section 3 we first conduct a survey to answer the research question **RQ**$_1$**:**_"How do software testing professionals perceive and evaluate the manual tests created by ChatGPT while comparing them to human-made tests?"_ and perform a qualitative analysis. The survey received answers from 18 testing professionals, which compared the differences between a human-made and an LLM-made test, selecting the best one in two particular scopes: "Completeness" and "Clarity". The respondents found that the ChatGPT tests are more complete than the human tests by a difference of 5.56%, while the human tests are written more clearly by a difference of 6.95%. In section 4 we discuss in a detailed manner when each technique tends to perform better.

Second, we execute a quantitative analysis by running similarity comparison algorithms between same application tests, utilizing metrics such as TF-IDF and ROUGE-L. Here we focus on the research question **RQ**$_2$**:**_"How do ChatGPT-generated and human-authored test cases compare in terms of semantic similarity and structural quality in automated similarity metrics?"_. After comparing the 12 pairs of natural language tests, we found that one third of the ChatGPT-generated tests shared over 50% similarity with human-authored tests.

In summary, this paper provides the following contributions:

- A tool to generate manual tests using ChatGPT's API. (Section 3.1);
- A survey with software testing professionals to validate the tests produced by the tool (Section 3.3);
- A quantitative study to check wether LLM-generated tests are similar in terms of semantics, structure and wording to human-made tests (Section 3.4).

A replication package with all results is available at Figshare [38]. Our tool is available at our companion website [3].

2 Natural Language Tests

In the context of software engineering, the quality of the application is vital, as lack of quality can lead to disasters, _e.g._ rocket explosions [10], money loss [26], and service outage [11]. These problems can be mitigated by applying software testing, be it of any kind. Natural Language Tests, which describe test scenarios in human-readable formats, serve as a bridge between technical

Test name: `firefox/firefox-006`

Preconditions: `This test will check that Firefox can print websites.`

#	Actions	Verifications
...	...	...
3	`Select "print to file" as printer and enter "firefox.pdf" as filename. Select your home folder as location. Then click on "Print"`	`A window opens, showing the progress of the print`
...	...	...

Figure 2: Test Case from Ubuntu OS

requirements and actual user experiences, making them indispensable for validating software functionality from a user's perspective. As manual testing techniques are to be understood and executed by testers with varying levels of technical expertise, they require in the simplest case only a readable sheet of paper and a working system. Because of this, manual tests remain critical in the context of fast-paced, ever-changing software.

Consider the example of a manual test developed by the contributors to Ubuntu OS [39] seen in Figure 2. The test outlines specific steps for validating a particular functionality within the Ubuntu operating system's graphical user interface in a specific fashion. The test includes detailed instructions that guide a tester through a sequence of actions and verification points.

The writing of comprehensive test cases poses a challenge to quality assurance teams, because testing activities account for 30 to 90 percent of labor expended to produce a working program [13]. The cost implications of manual test creation extend beyond theoretical concerns. A comprehensive industry survey by Capgemini [9] revealed that organizations spend an average of 23-35% of their IT budgets on quality assurance activities, with test case creation and maintenance representing a substantial portion of this expenditure. Furthermore, the same study indicated that 44% of surveyed organizations identified test case creation as a "significant bottleneck" in their development processes.

In a large software as Ubuntu, writing manual tests are not priority, as evidenced by the spacing of 3 to 4 days between commits on the manual testing repository. This time-range can be gathered by running simple Git [2] commands such as `git rev-list --reverse HEAD` and counting the differences between the timestamps. As writing tests in natural language is very demanding, this may be one of main causes of this delayed update. But this problem is not isolated to large software, as the optimization of this process is studied in many areas such as requirements [21, 40, 41] and test automatization [17, 22, 27, 29, 42].

Previous works try to alleviate this problem leveraging frameworks to automatize the creation and execution of tests, by employing Gherkin [25] or Selenium [15], but while quickening the process, the operator still needs to spend some time, be it by interacting with the system or adjusting code.

While manual testing has been extensively studied in prior research [12, 23, 24, 31, 32, 36, 37], existing literature primarily focuses on test analysis and maintenance rather than test generation. The majority of previous work addresses natural language test smells — indicators of structural or content-related deficiencies within test specifications — while other studies concentrate on test cataloging and defect removal. However, none of these investigations tackle the fundamental challenge of automated manual test production, representing a significant gap in the current body of knowledge.

On the other hand, the advent of large language models (LLMs) decrease resource drainage and increase time effectiveness. As test cases are written in natural language, ChatGPT may pose a viable solution to this. The resource intensity, i.e., low time-resource usage, of natural language testing has spurred interest in AI solutions. Large language models like ChatGPT offer new possibilities for cost reduction while preserving test quality.

Given ChatGPT's contextual understanding and text generation capabilities, we investigate whether AI systems can lower test creation costs without compromising quality. This exploration could reshape testing efficiency and resource allocation in quality assurance.

3 Empirical Study

This section presents a study divided into a qualitative analysis, conducted as an online survey with software testing professionals, and a quantitative analysis, conducted as a similarity inspection using different algorithms. Here, we aim to provide data to answer the following research questions:

- **RQ$_1$: "How do software testing professionals perceive and evaluate the manual tests created by ChatGPT while comparing them to human-made tests?"**
- **RQ$_2$: "How do ChatGPT-generated and human-authored test cases compare in terms of semantic similarity and structural quality in automated similarity metrics?"**

The objective of this study is to evaluate ChatGPT's capability in generating manual tests that meet human acceptability standards. To achieve this goal, we systematically assess the opinions of software testing professionals regarding AI-generated tests through a controlled comparison methodology. Specifically, we present participants with side-by-side comparisons of human-created tests versus LLM-generated tests, ensuring that the professionals remain unaware of each test's origin to eliminate potential bias. Through comprehensive analysis of participant responses to both "Test A" and "Test B" conditions, combined with detailed examination of their qualitative comments and feedback, we aim to determine whether respondents can discern meaningful differences between human and AI-generated testing materials. This analysis serves to validate the practical acceptability of ChatGPT's test creation capabilities in professional software testing contexts.

3.1 Settings and Data Acquisition

This study begins by selecting a sample of systems that can be categorized into categories and taking screenshots of those systems. To facilitate data acquisition and enable reproducibility, all systems should be widely accessed websites. To increase fairness in analysis and for the LLM to provide better answers, all websites must also be in English and provide many features.

To increase the generality of our study, we selected four distinct categories of systems, all websites: *E-commerce*, *E-mail*, *Maps*, and *Search*. For each category, three different websites were chosen, totaling 12 websites in our evaluation. Those websites were collected from SimilarWeb top websites ranking [8] for each category. These different categories, while not enforcing full comprehension of all types of systems, can provide some insights about different systems. The selected model was *GPT-4o*, using OpenAI's API [4]. To select the websites for our study, we prioritized platforms that are widely recognized for their popularity and credibility. Additionally, we ensured that at least one member of our team had prior experience using the chosen website, enabling us to confidently validate the test cases presented in subsequent sections. This approach strengthens the reliability and reproducibility of our results.

3.2 Procedure

For each website, one screenshot was taken. Using these screenshots, three different researchers elected objectives for each test, while trying to simulate the process of natural language test creation. The list of objectives is available at Listing 1. The objectives are purposefully simple and easily done by a human. With this decision, the results are less prone to subjectivity and can be reproduced more easily by other researchers.

Listing 1: Test Objectives

```
E-commerce
    - Aliexpress: login with Google
    - Amazon US: go to the Returns & Orders menu
    - Walmart- Search for a Playstation 5

Emails
    - Gmail: use the "Compose" button to create a new
     email
    - Outlook: mark all emails as read
    - Yahoo!Mail: archive an email

Maps
    - Bing Maps: get directions to UFAL destination
    - Google Maps: search for restaurants using the
     Restaurants button
    - Waze: click on the police icon and see if it works

Search
    - Google Search: reverse search using the pale blue
     dot image
    - Bing Search: search for the pale blue dot image
    - Yahoo Search: search for the pale blue dot image
     using voice search
```

These objectives are then passed as input to both ChatGPT-4o and a testing professional with more than six years of experience who is not an author of this work. The overview of this procedure is shown in Figure 3. The researcher's role is limited to capturing and defining the test objectives. Both the language model and the human expert receive the same inputs: the objective and the screenshot.

To reduce bias and improve reproducibility, the prompt leverages several established patterns from the prompt engineering catalog [43]: the Personas pattern, sequential instructions (Recipe pattern), format constraints (Template pattern), and objective-screenshot alignment (Context Control). The model is explicitly assigned a role, instructed to follow a sequence of ordered steps, and required to produce output in a structured table with fixed columns and optional preconditions. Each response is guided by a clearly defined objective. The complete prompt is provided in Listing 2. Notably, the objective must always be updated according to each new screenshot,

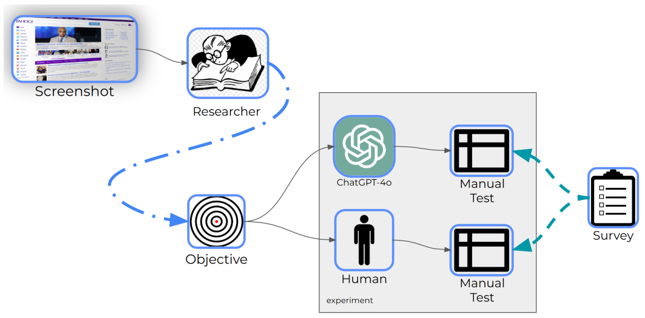


Figure 3: Qualitative Analysis Methodology

ensuring that the objective-screenshot pair remains consistent and contextually aligned throughout the process.

The prompt intentionally provides only minimal context (the objective and the screenshot) to establish a conservative, lower-bound benchmark for LLM performance. This design choice ensures that any measured effectiveness reflects baseline capabilities, which could be improved with richer contextual information.

Listing 2: Test Case Prompt Instructions

```
[SYSTEM]: You are a specialist in manual natural language
      tests. You have to test a specific described
     functionality. Follow a professional and clear style
      guide. First step is to analyze the picture, then
     write the test cases. Each test should have at least
      2 steps.

[USER]: Create a manual test case suite with 1 test cases
      for the provided screenshot, where the test will
     have the single objective: **OBJECTIVE**. Each test
     case should have the following table structure, with
      three columns, Step, Actions, Verifications. Also,
     add a title to each table. Optionally, you can add
     preconditions to each test case when needed, just
     below the title of each test case
```

In total, we achieved 12 pairs (GPT and Human) of tests. These tests are also available at our Figshare replication package [38]. It is important to note that all tests were inspected by the authors to ensure they were executable; neither ChatGPT nor the human participants produced commands that were impossible to follow.

To create the tests, we used OpenAI's API with default settings (e.g., temperature, top-p) and a standard prompt (Listing 2) paired with a raw screenshot. No visual grounding or image element tagging was used. For a fair evaluation, we accepted all ChatGPT outputs without filtering or regenerating low-quality responses. Since the objectives were simple, the model accurately generated the tests without any hallucinations. This may not be the case for more complex interfaces that require multiple steps or actions, which would require more pictures, i.e., more context.

3.3 Qualitative Analysis - Survey

Using the outputted tests, we created three different surveys, which had four sections each, one for each category. In a measure to mitigate the strain bias, after every answer submission these sections were scrambled using Google Script [1]. Participants for the survey were recruited through social media and email invitations. All responses were collected anonymously, and participants were informed that their data would remain confidential.

3.3.1 Format All surveys' sections have the same format and structure - first, we show the screenshot and the objective selected for this specific test. Second, human and LLM tests are shown side by side, named Test A and Test B. All human-made tests are labeled *Test A*, while GPT-made tests are labeled *Test B*. Third, the participant is asked to answer two different questions:

- Which test covers all necessary functional aspects?
- Which test do you believe provides clearer instructions?

The idea behind those questions is to understand alignment with objectives, which we shall be called "Completeness" and clarity in the writing, which we shall be called "Clarity". These concepts will appear in further subsections.

The surveyed could then select one between *Test A, Test B, Neither, Both Equally*. There was also an open text field for comments. Some social aspects of the respondent were additionally collected, such as their principal field of occupation, years of experience and location. Besides, to increase generality, the three surveys were scrambled using a redirect scramble tool called allocate monster [20], which after every click on the link would randomly select among the three available Google Forms.

3.3.2 Results The survey ran during 2 months (January and February of 2025), collecting 18 answers. To "*What is your principal occupation?*", the answers were equally balanced, 50% responded *Industry* and the rest responded *Academy*. As per years of experience with software testing, the surveyed had an average of 3.11 years, with 27.8% of the respondents with 5 or more years, and 5.6% of the respondents with less than one year. We had 50% of answers from Brazil, 44.4% from Italy and 5.6% from Belgium.

Regarding the tests, the condensed answers of the survey is shown in Table 1, while the detailed results are shown in Table 2, which separates the answers per application. On Table 2, as only the Google Maps had a *Neither* on the "Completeness" aspect, we suppressed this value due to size limitations.

Table 1: Condensed Answers of Survey

Test - Metric	Test A	Test B	Both	Neither
Email - Completeness	**9**	4	5	-
Email - Clarity	**11**	3	4	-
Maps - Completeness	5	5	**7**	1
Maps - Clarity	5	**8**	5	-
Search - Completeness	3	**9**	6	-
Search - Clarity	6	**7**	5	-
Ecommerce - Completeness	5	**8**	5	-
Ecommerce - Clarity	**8**	7	3	-
Total Completeness	22 (30.56%)	**26 (36.11%)**	23 (31.94%)	1 (1.39%)
Total Clarity	**30 (41.67%)**	25 (34.72%)	17 (23.61%)	0 (0.00%)
TOTAL COMBINED	**52 (36.11%)**	51 (35.42%)	42 (29.17%)	1 (0.69%)

Observing Table 1, a notable difference appears when the answers are compared by domain of application. For instance, Test A, i.e. the human-made, is clearly more effective in both clarity and completeness in the Email domain; Test B, i.e. the LLM-made, is superior in both on the Search domain. This may be due to the increasing level of text present on the screen. The Email domain is the one that has the most text showed upfront (and mostly disconnected sentences), while Search has the least text showed on screen.

Overall, **the human-made tests had the most combined acceptance rate** (36.11%) **with prevalence on the clarity aspect**, as shown by Table 1 on the TOTAL COMBINED row. The percentages on this table are based on the total of answers for each question, namely 72 answers for Completeness' question and another 72 answers for Clarity's question, totalizing 144 answers.

Still, Table 1 may hide critical details that are not available when the answers are pictured as a domain-level analysis. Table 2 shows an application-level analysis, which depicts Aliexpress, a case that will be explored further. It is important to note that the percentages reported in this table are calculated based on the total number of responses received for each specific application under evaluation. For instance, the Amazon application received six total responses, and since Test A received four positive evaluations in the Completeness category, this resulted in a 66.67% acceptance rate for that particular application-metric-test combination.

Upon inspection of Table 2, the primary conclusion is that neither Test can be considered universally superior. While Test A tends to provide clearer results (41.67% acceptance rate), Test B tends to be more complete (36.11%). The best choice is highly dependent on the specific application domain and which metric is prioritized, be it Clarity or Completeness. However, the differences in the totals are relatively small (6.95% for Clarity and 5.56% for Completeness) and the "Both" column in Completeness shows that nearly a third of answers found tests to be equally complete.

Regarding Aliexpress on Table 2, we can see the trade-off between completeness and clarity. While Test A has 0% acceptance for completeness, it has 60% acceptance for clarity. Conversely, Test B has 80% acceptance for completeness and 0% for clarity. This application highlights a direct trade-off between the two metrics, which suggests that the LLM could finish the task but failed to present it, while the human functioned on the contrary.

Therefore, the choice between the two approaches is not just a matter of preference but a strategic decision based on whether the primary goal is to achieve the objective or have a sound presentation.

Results from Table 1 show that users rated Test A for clarity and Test B for completeness. Feedback reveals a story behind these scores. Test B's rating in completeness failed to translate into user preference; users judged its design as "worse" and "fragile to changing implementations". Test A won on the metric of clarity, and user comments suggest this resulted from its thoroughness: "very clear", "too detailed in certain aspects". Participants praised Test A for being "more detailed" and for avoiding the "risk of being done incorrectly". Thus, users prioritized the clarity from Test A's design, even if it meant Test B scored higher on the isolated metric of Completeness.

Ultimately, the characteristics praised on Test A are essentially Completeness features. This suggests that the users perceived the Clarity metric as one that emerges from the Completeness. Therefore, this analysis suggests that while Clarity can exist in isolation, the type of Clarity users prioritize emerges from a foundation of Completeness.

Table 2: Application-wise Absolute and Relative Results

Application	Completeness			Clarity		
	Test A	Test B	Both	Test A	Test B	Both
Outlook	1 (20%)	**2 (40%)**	2 (40%)	1 (20%)	1 (20%)	**3 (60%)**
Gmail	**5 (72%)**	1 (14%)	1 (14%)	**6 (86%)**	1 (14%)	0
Yahoo! Mail	**3 (50%)**	1 (17%)	2 (33%)	**4 (66%)**	1 (17%)	1 (17%)
Waze	**2 (40%)**	1 (20%)	2 (40%)	**3 (60%)**	0	2 (40%)
Google Maps	0	**3 (43%)**	3 (43%)	0	**6 (86%)**	1 (14%)
Bing Maps	**3 (50%)**	1 (17%)	2 (33%)	2 (33%)	2 (33%)	2 (33%)
Yahoo! Search	2 (40%)	2 (40%)	1 (20%)	**4 (80%)**	0	1 (20%)
Bing Search	0	**4 (57%)**	3 (43%)	0	**4 (57%)**	3 (43%)
Google Search	1 (17%)	**3 (50%)**	2 (33%)	2 (29%)	**3 (43%)**	1 (17%)
Amazon	**4 (66%)**	1 (17%)	1 (17%)	**3 (50%)**	2 (33%)	1 (17%)
Aliexpress	0	**4 (80%)**	1 (20%)	**3 (60%)**	0	2 (40%)
Walmart	1 (14%)	**3 (43%)**	3 (43%)	2 (40%)	**5 (72%)**	0
TOTAL	22	**26**	23	**30**	25	17
TOTAL %	30.55%	**36.11%**	31.94%	**41.67%**	34.72%	23.61%

3.4 Quantitative Analysis - Similarity

To conduct the quantitative analysis, this paper utilizes a Streamlit [6] application to perform a similarity comparison between the two tests that have the same objective, inspecting correlations between both their actions and verifications. The approach involves the use of natural language processing (NLP) techniques to evaluate the semantic similarity between tests. The script used can be found on our companion website [3].

3.4.1 Similarity Calculation These approaches are used for similarity comparison:

- **Cosine Similarity with TF-IDF**: The TF-IDF [35] vectorizer is used to transform actions and verifications into feature vectors, calculating the cosine similarity between those from one test to another, e.g. the distance between Test A's action to Test B's same step action and henceforth.
- **Semantic Similarity with BERT**: The Sentence-BERT [34] model ‘paraphrase-MiniLM-L6-v2‘ is applied to understand the semantic meaning of actions and verifications, calculating the similarity between the generated embeddings from one test to another.
- **ROUGE-L**: The ROUGE-L [28] metric calculates the longest common subsequence (LCS) between texts to evaluate structural overlap, combining recall (coverage of reference content), precision (relevance of generated content), and their harmonic mean (F1-score). It measures sequential word matches while tolerating minor wording variations, making it robust for comparing actions/verifications between tests where syntactic fidelity matters more than semantic equivalence.

Although these metrics account for similarities, each approach has a specific focus and objective in this study, which can be seen in Table 3. The metrics being combined are capable of expressing a broad similarity score.

Yet, there are cases where the steps length are not the same, therefore we use the following algorithm to handle these cases:

- If both values are null, the similarity is considered 1.0 (100% similar).

Table 3: Metrics Used

Metric	Focus	Objective in the Study
TF-IDF	Lexical frequency	Detection of key terms
BERT	Global semantics	Validates conceptual equivalence
ROUGE-L	Execution order	Guarantees logical flow integrity

- If only one value is null, the similarity is 0.0 (0% similar).
- Otherwise, similarity is calculated based on the embeddings generated by BERT.

The algorithm handles edge cases while maintaining semantic accuracy for content comparisons. For better understanding, there is a pseudo-code implementation in Listing 3.

Listing 3: Similarity Calculation Algorithm Pseudo-code

```
function calculateBERTSimilarity(value1, value2):
    if value1 is null and value2 is null:
        return 1.0  // 100% similar
    else if value1 is null or value2 is null:
        return 0.0  // 0% similar
    else:
        embedding1 = BERT.generateEmbedding(value1)
        embedding2 = BERT.generateEmbedding(value2)
        similarity = computeSimilarity(embedding1,
     embedding2)
        return similarity
```

3.4.2 Comparison and Visualization The comparison is performed row by row between two tests with the same objective, calculating semantic similarity for each pair of actions and verifications, while also capturing Rouge-L and TF-IDF metrics. The Streamlit application offers an interactive interface where the user can:

- Generate an overall similarity comparison between test types for each named test, visualizing the results in a horizontal bar chart with gradient colors to indicate similarity intensity.
- Select a specific test for a detailed comparison between two test types, displaying metrics such as average, minimum, maximum similarity, and the number of rows compared, along with the data for each test.

To quantify similarity between rows across test cases, we compute a row score by averaging the similarity values of their components. Row-level similarity assessment involves calculating the mean of action similarity and verification similarity scores, as formalized in the pseudo-code presented in Listing 4. The similarity calculation framework operates according to the methodology:

- **Action Similarity**: Calculated using BERT (scores ranging from 0.0 to 1.0)
- **Verification Similarity**: Calculated using BERT (scores ranging from 0.0 to 1.0)
- **Row Score**: Calculated as the mean of action and verification similarities: (Action Similarity + Verification Similarity) / 2
- **Test Similarity Score**: Row Score / Number of Rows

Listing 4: Test Similarity Calculation Algorithm Pseudo-code

```
function calculateTestSimilarity(testA, testB):
    totalScore = 0
    numberOfRows = length(testA)
    for i = 0 to numberOfRows - 1:
        actions = calculateBERTSimilarity(testA[i].action
      , testB[i].verification)

        verifications = calculateBERTSimilarity(testA[i].
     verification, testB[i].verification)

        rowScore = (actions + verifications) / 2

        totalScore = totalScore + rowScore

    testScore = totalScore / numberOfRows
    return testScore
```

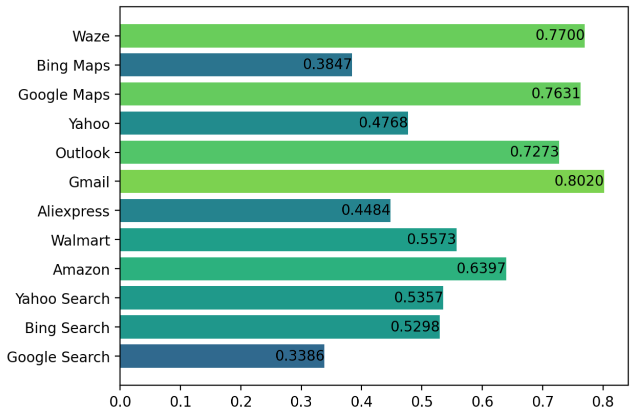


Figure 4: Average BERT Similarity for Each Test

This framework demonstrates flexibility and is adapted throughout the code to different similarity measurement approaches, including BERT-based similarity, TF-IDF vectorization, or ROUGE-L analysis. The design enables evaluation across similarity metrics while maintaining scoring methodology.

3.4.3 Results The BERT comparison, presented in Figure 4, reveals the average semantical similarity between test types for each named test. The horizontal bar chart visualization allows for a quick understanding of consistency between tests, with less intense colors indicating higher similarity. The produced graph is shown in Figure 4. This graph is extracted from the "Average Similarity %" column presented in the Table 5.

We had a total semantical average similarity of 58.11%, minimum semantical average similarity of 33.86% (Google Search) and maximum semantical average similarity of 80.20% (Gmail). The Google Search tests are shown in Figure 7 and the Gmail tests are displayed on Figure 8.

Table 5 presents a deep analysis about each row score, showing average, maximum and minimum similarities in percentages and the overall row score. The "Compared %" column displays the percentage of rows that have been compared. Also, the column "Rows Diff" shows the difference in number of rows. This difference is calculated subtracting the number of rows present in Test A from the number of rows present in Test B, namely:

$$nRowsB - nRowsA = rowsDiff \quad (1)$$

Table 4: All Similarities Results

Test Name	TF-IDF Similarity	BERT Similarity	ROUGE-L Score	Rows Diff
Google Search	0.0871	0.3386	0.1263	1
Bing Search	0.2763	0.5298	0.1703	0
Yahoo Search	0.2579	0.5357	0.2028	-2
Amazon	0.4915	0.6397	0.5503	-5
Walmart	0.3155	0.5573	0.2720	1
Aliexpress	0.1714	0.4484	0.1706	-4
Gmail	0.4533	0.8020	0.5051	-5
Outlook	0.4542	0.7273	0.5536	0
Yahoo	0.3010	0.4768	0.1574	-4
Google Maps	0.3666	0.7631	0.3673	0
Bing Maps	0.1833	0.3847	0.1513	-2
Waze	0.5737	0.7700	0.4298	-1
TOTAL AVERAGE	0.3277	0.5811	0.3047	-2

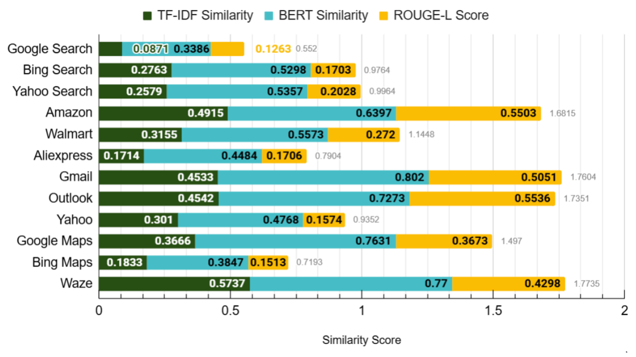


Figure 5: All Similarities Stacked

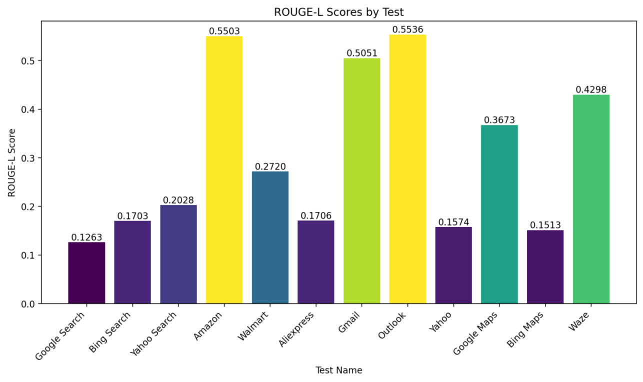
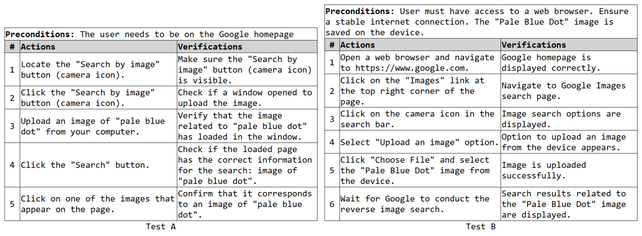


Figure 6: Rouge-L Scores by Test

Therefore, if Test B has more rows, the number is greater than zero. If the number is lower than zero, than Test A has more rows. It equals to zero if the number of rows between both tests is equal.

Furthermore, the results obtained from Rouge-L are presented at Figure 6. Still, the evaluation with all methods is available at Table 4. In this table we can see that the similarity score pattern is almost exactly followed by each method, i.e, the more similar one is on TF-IDF, more it will be on BERT and ROUGE-L. This is evident in Figure 5, which stacks each similarity metric, presenting a visual comparison of each score.



Preconditions: The user needs to be on the Google homepage

#	Actions	Verifications
1	Locate the "Search by image" button (camera icon).	Make sure the "Search by image" button (camera icon) is visible.
2	Click the "Search by image" button (camera icon).	Check if a window opened to upload the image.
3	Upload an image of "pale blue dot" from your computer.	Verify that the image related to "pale blue dot" has loaded in the window.
4	Click the "Search" button.	Check if the loaded page has the correct information for the search: image of "pale blue dot".
5	Click on one of the images that appear on the page.	Confirm that it corresponds to an image of "pale blue dot".

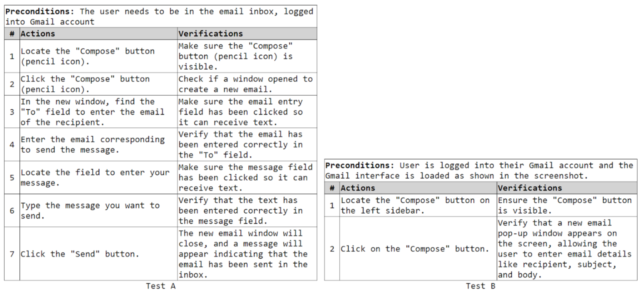
Test A

Preconditions: User must have access to a web browser. Ensure a stable internet connection. The "Pale Blue Dot" image is saved on the device.

#	Actions	Verifications
1	Open a web browser and navigate to https://www.google.com.	Google homepage is displayed correctly.
2	Click on the "Images" link at the top right corner of the page.	Navigate to Google Images search page.
3	Click on the camera icon in the search bar.	Image search options are displayed.
4	Select "Upload an image" option.	Option to upload an image from the device appears.
5	Click "Choose File" and select the "Pale Blue Dot" image from the device.	Image is uploaded successfully.
6	Wait for Google to conduct the reverse image search.	Search results related to the "Pale Blue Dot" image are displayed.

Test B

Figure 7: Google Search Tests - "reverse search using the pale blue dot image"



Preconditions: The user needs to be in the email inbox, logged into Gmail account

#	Actions	Verifications
1	Locate the "Compose" button (pencil icon).	Make sure the "Compose" button (pencil icon) is visible.
2	Click the "Compose" button (pencil icon).	Check if a window opened to create a new email.
3	In the new window, find the "To" field to enter the email of the recipient.	Make sure the email entry field has been clicked so it can receive text.
4	Enter the email corresponding to send the message.	Verify that the email has been entered correctly in the "To" field.
5	Locate the field to enter your message.	Make sure the message field has been clicked so it can receive text.
6	Type the message you want to send.	Verify that the text has been entered correctly in the message field.
7	Click the "Send" button.	The new email window will close, and a message will appear indicating that the email has been sent in the inbox.

Test A

Preconditions: User is logged into their Gmail account and the Gmail interface is loaded as shown in the screenshot.

#	Actions	Verifications
1	Locate the "Compose" button on the left sidebar.	Ensure the "Compose" button is visible.
2	Click on the "Compose" button.	Verify that a new email pop-up window appears on the screen, allowing the user to enter email details like recipient, subject, and body.

Test B

Figure 8: Gmail Tests - "use the "Compose" button to create a new email"

Table 5: Row BERT-Similarities Between Test A and B

Applications	Average Similarity %	Minimum Similarity %	Maximum Similarity %	Compared %	Rows Diff
Google Search	33,86	25,92	46,82	83,33	1
Bing Search	52,98	44,84	63,12	100,00	0
Yahoo Search	53,57	42,68	81,94	66,67	-2
Amazon	63,97	62,92	65,03	28,57	-5
Walmart	55,73	29,21	75,49	80,00	1
Aliexpress	44,84	34,44	55,89	50,00	-4
Gmail	80,20	80,11	80,29	28,57	-5
Outlook	72,73	70,99	74,47	100,00	0
Yahoo Mail	47,68	37,06	58,30	33,33	-4
Google Maps	76,31	63,82	87,00	100,00	0
Bing Maps	38,47	24,22	69,81	66,67	-2
Waze	77,00	76,91	77,09	66,67	-1
TOTAL AVERAGE	**58,11 %**	**49,43 %**	**69,60 %**	**66,98 %**	**-2**

4 Discussion

This section presents a discussion about the study, conducted as an online survey with software testing professionals. Here we aim to answer both research questions while generating insights from both analysis.

4.1 About the Qualitative Analysis

Analysis of the results in Table 2 reveals that **human-authored tests (30.55%) were perceived as less complete than those generated by ChatGPT (36.11%)**. This outcome can likely be attributed to the nature of LLMs, which are optimized to follow instructions and fulfill defined objectives with precision. Consequently, LLMs appear to adhere more strictly to initial objectives than human authors. This suggests a significant practical benefit for early-stage test creation; using an LLM to generate a test suite from scratch could accelerate the process by providing a strong initial scaffold that is closely aligned with project requirements.

On the Clarity field, **the humans' tests (41.67%) are considered clearer than ChatGPT's tests (31.94%)**. An explanation for this is rooted in how LLMs generate text. While they can be direct, their probabilistic, next-word-prediction process can sometimes fail to maintain a coherent chain-of-thought from a human perspective. This can result in phrasing that introduces subtle confusion for the person executing the test, thus reducing perceived clarity.

Furthermore, the high agreement in the "Both" column for Completeness suggests that the distinction between the two tests was not critical, and both produced complete tests. This indicates that the **practical differences in Completeness may be less impactful than the differences in Clarity**.

Moreover, the difference between Test A and Test B in the Clarity field is more pronounced than the difference on the Completeness field ($6.95 > 5.56$). This is evidence of how much clearer the instructions provided by the humans are when compared with the ones provided by ChatGPT.

Figure 9 illustrates an analogy for this behavior pattern. Consider navigating from point A to point B. In our visual experiment, human problem-solving resembles a direct straight-forward efficient path but often terminates in proximity to the target without precisely reaching it (Human -> nearB). In contrast, ChatGPT's approach follows a more sinuous route, exploring intermediate points and making adjustments along the way, but ultimately converges exactly at the intended destination (ChatGPT -> B).

This configuration may have some outcomes that can enhance the process of creation of natural language tests. We can leverage the completeness of LLMs and the clarity of humans. The usage of ChatGPT as a scaffold to writing objective-aligned, while providing acceleration to the creation, may leave the Q&A team confused with the unclear instructions. Therefore, a mix between ChatGPT and humans should be more accepted in the generation of manual tests, with the human re-writing or changing the wording of the generated test, to adequate it to a better text flow.

Answer to RQ_1**.** The online survey results indicate that software testing professionals tend to perceive ChatGPT-generated tests as more complete, while human-authored tests are viewed as clearer.

4.2 About the Quantitative Analysis

Our analysis draws primarily from Figure 4 and Table 5, which present key metrics on test similarities.

The Gmail case shows a notably high average similarity of 80.20% despite comparing only 28.57% of rows, representing a difference of 5 fewer rows in the LLM-generated test compared to the human-written one. This limited comparison base likely contributes to the elevated similarity score.

However, this explanation does not hold for the Aliexpress case, which shows a moderate similarity (approximately 45%) despite having a similar deficit of 4 rows.

As both are of similar level of complexity, this deviation likely occurs because Aliexpress' objective is fairly concise between all platforms (login with Google), i.e is widely available on the data that the GPT has been trained upon, whereas Gmail's objective (use the "Compose" button to create a new email) is domain-specific and therefore not so much available on the training dataset. Therefore, platform-specific characteristics significantly impact similarity outcomes.

Table 5 reveals that approximately 67% of all rows were successfully compared across test cases, yielding an **overall average semantical similarity of approximately 58%**. The general pattern shows ChatGPT generating tests with an average of two fewer rows than human testers. This finding appears to contradict the pattern illustrated in Figure 9, where ChatGPT is perceived as having greater "Completeness" than humans. This apparent contradiction can be resolved by recognizing that the completeness metric specifically measures achievement of proposed objectives rather than verbosity. Thus, while ChatGPT produces more concise test descriptions with fewer rows, these descriptions may sacrifice clarity in the process.

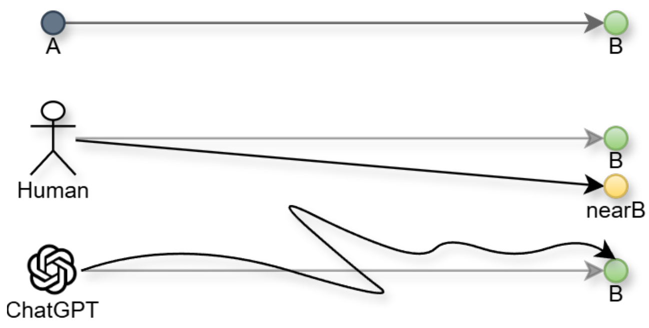


Figure 9: Car Route Analogy

From Figure 5 and Table 4, the analysis denotes a notable discrepancy between structural and semantic similarity metrics. While **structural similarity measures only 31.62%**, indicating significant divergence in test organization and step sequence, **semantic similarity reaches 58.11%**, demonstrating substantial alignment in test intent and functional coverage. This contrast suggests that while ChatGPT-generated tests diverge from human-authored tests in their structural formulation, they maintain comparable semantic equivalence in terms of test purpose and requirement fulfillment, i.e. the test steps are differently organized from the human-made tests but they maintain functional coverage with alternative steps.

Answer to RQ_2**.** Although semantically similar (58.11%), the structural similarity of ChatGPT-generated tests measures only 31.62%. This discrepancy of 26.49% suggests that LLMs can achieve functional coverage through alternative test steps.

5 Threats to Validity

5.1 Internal Threats

We identified two internal threats: *selection bias* in Objectives and Websites. Objective selection bias may occur if the chosen tasks do not cover the range of relevant user actions, which could limit the scope of our findings. Website selection bias may arise if the selected sites do not reflect common usage patterns, reducing the study's representativeness.

To address these risks, we discussed objective selection with another research group and used an online visitation ranking to choose websites [8]. These steps help reduce bias, but some may remain. The results may still be influenced by the chosen objectives and websites. Expanding the diversity of both in future work could improve the robustness and generalizability of our conclusions.

5.2 External Threats

We identify two main external threats to this study. The first is the *low response rate* to the questionnaire, which may limit the generalization of the findings. To mitigate this, we sought participants with diverse backgrounds and from a variety of locations, aiming for a broader representation. The second threat is *response fatigue*, where participants may become tired while answering the questionnaire, potentially affecting the quality of their responses. This was addressed by dividing the questionnaire into smaller sections and randomizing the order of questions to help maintain participant engagement.

5.3 Construct Threats

Two construct threats were also identified. The first concerns the *variability of model responses*, as the model may not always produce accurate or consistent answers, which could hinder reproducibility. To address this, we used standard configurations and parameters to minimize model inconsistent behavior throughout the experiments. The second threat is the *bias in manual test creation*, since the manual test depends on the subjective judgment of its creator, potentially introducing measurement bias. This was mitigated by involving multiple independent reviewers and applying a standardized validation checklist during the test development process.

6 Related Work

As the field of manual testing continues to mature, there is a notable absence of studies focused exclusively on the generation of natural language test cases, particularly using the layout and methodology proposed in this paper. While numerous works [14, 19, 21, 40, 41, 45] leverage LLMs or NLP techniques to generate test cases from various forms of input (requirements, user stories, or code), these typically aim to produce automated tests rather than human-readable test instructions. In this sense, they share a general objective with our approach: transforming structured or semi-structured input into test cases. However, they diverge significantly in either the input modality, the generation method, or the form of the output, which in our case is explicitly designed for manual, human-driven testing workflows.

Rane *et al.* [33] aimed to automate the generation of test cases from Agile user stories written in natural language. Their approach differs significantly from ours, which relies on screenshots and predefined testing objectives. While our method focuses primarily on visual elements and the graphical interface to infer user behavior and application state, Rane's work emphasizes understanding and processing textual requirements. By leveraging a test scenario description and a domain-specific dictionary, their system uses natural language processing to generate functional test cases, addressing the challenges of requirement ambiguity and frequent changes typical in Agile development.

Ouedraogo *et al.* [30] focused on evaluating the effectiveness of Large Language Models (LLMs) in generating unit test code for existing Java classes. In contrast, our work does not involve generating executable code, but rather produces natural language test cases based on high-level inputs such as application screenshots and manually defined test objectives. This fundamental difference in both input and output shifts our focus from code-level validation to human-guided exploratory testing.

Nebut *et al.* [29] aimed to generate test scenarios from use case scenarios and machine-derived test objectives. Our framework differs from theirs in both input modality and objective specification. Rather than extracting use cases from textual requirements, we provide an LLM with a screenshot of the application interface. Additionally, the test objective in our approach is not automatically generated but manually defined by a human, allowing for more intentional and context-aware test case generation.

Zhao *et al.* [46] generated tests from a video of an actor utilizing the UI of some application. Their Avgust approach processes videos through computer vision and natural language processing to detect user actions, builds a state machine-based model, and generates tests for target applications. While Avgust uses visual input like our method, it differs in approach and output: Avgust processes video content to produce executable tests, whereas our method uses screenshots to create natural language test cases for manual testing. Avgust's model targets cross-application test generation, while our approach creates test instructions for the application interface shown in the screenshot.

7 Concluding Remarks

In this paper, we introduced a tool for generating natural language test cases from screenshots and objectives, evaluating the quality of generated tests through a comprehensive assessment involving 18 software testing professionals and a quantitative analysis of generated versus human-authored tests. The evaluation revealed complementary strengths: professionals rated human-authored tests as clearer while perceiving ChatGPT-generated tests as more complete. Quantitative analysis demonstrated that while tests exhibit semantic similarity (58.11%), they diverge significantly in structural organization (31.62%), indicating that LLMs achieve functional coverage through alternative test design patterns. As future work, we intend to:

- Explore test case generation capabilities with alternative large language models (e.g., Claude, Gemini) to assess generalization across different architectures;
- Incorporate automated test objective selection directly from screenshot analysis, reducing manual specification requirements;
- Investigate hybrid approaches combining human expertise with LLM capabilities to optimize both clarity and completeness;
- Extend evaluation to additional domains and application types to validate findings across diverse software contexts.

ACKNOWLEDGMENTS

This work was partially funded by CNPq 312195/2021-4, 310313/2022-8, 403361/2023-0, 443393/2023-0, 404825/2023-0, 315840/2023-4, FAPEAL 60030.0000000462/2020, 60030.0000000161/2022, and FAPESB PIE002/2022 grants.

REFERENCES

[1] [n. d.]. Forms Service. https://developers.google.com/apps-script/reference/forms. [Accessed 15-06-2025].

[2] [n. d.]. Git - git-rev-list Documentation — git-scm.com. https://git-scm.com/docs/git-rev-list. [Accessed 15-06-2025].

[3] [n. d.]. Image2Test: Using ChatGPT to Build Manual Tests from Screenshots. https://github.com/easy-software-ufal/manual-test-generation

[4] [n. d.]. OpenAI Platform — platform.openai.com. https://platform.openai.com/docs/api-reference/chat/create?lang=python. [Accessed 15-06-2025].

[5] [n. d.]. State of Continuous Testing Report | BlazeMeter — perfecto.io. https://www.perfecto.io/resources/state-of-continuous-testing. [Accessed 10-02-2025].

[6] [n. d.]. *Streamlit Docs*. docs.streamlit.io

[7] [n. d.]. The State of Quality Report 2022. https://katalon.com/reports/state-quality-2022. [Accessed 16-06-2025].

[8] [n. d.]. Top Websites Ranking - Most Visited Websites in February 2025. similarweb.com/top-websites/.

[9] [n. d.]. World Quality Report 2023-24 — capgemini.com. https://www.capgemini.com/insights/research-library/world-quality-report-2023-24/. [Accessed 16-06-2025].

[10] Roni Amelan. 1996. Software testing blamed for Ariane failure. *Nature* 382, 6590 (01 Aug 1996), 386–386. https://doi.org/10.1038/382386a0

[11] Vasco Amorim, Armindo Fernandes, and Vitor Filipe. 2025. Analyzing the Impact of the CrowdStrike Tech Outage on Airport Operations and Future Resilience Strategies. *Procedia Computer Science* 256 (2025), 633–640.

[12] Manoel Aranda, Naelson Oliveira, Elvys Soares, Márcio Ribeiro, Davi Romão, Ullyanne Patriota, Rohit Gheyi, Emerson Souza, and Ivan Machado. 2024. A Catalog of Transformations to Remove Smells From Natural Language Tests. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (Salerno, Italy) *(EASE '24)*. Association for Computing Machinery, New York, NY, USA, 7–16. https://doi.org/10.1145/3661167.3661225

[13] Boris Beizer. 1990. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., USA.

[14] Shreya Bhatia, Tarushi Gandhi, Dhruv Kumar, and Pankaj Jalote. 2024. Unit Test Generation using Generative AI : A Comparative Performance Analysis of Autogeneration Tools. 54–61. https://doi.org/10.1145/3643795.3648396

[15] Andreas Bruns, Andreas Kornstadt, and Dennis Wichmann. 2009. Web application tests with selenium. *IEEE software* 26, 5 (2009), 88–91.

[16] José Campos, Andrea Arcuri, Gordon Fraser, and Rui Abreu. 2014. Continuous test generation: Enhancing continuous integration with automated test generation. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 55–66.

[17] Marcantonio Catelani, Lorenzo Ciani, Valeria L Scarano, and Alessandro Bacioccola. 2011. Software automated testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use. *Computer Standards & Interfaces* 33, 2 (2011), 152–158.

[18] Ting Chen, Xiao-song Zhang, Shi-ze Guo, Hong-yuan Li, and Yue Wu. 2013. State of the art: Dynamic symbolic execution for automated test generation. *Future Generation Computer Systems* 29, 7 (2013), 1758–1773.

[19] Arghavan Dakhel, Amin Nikanjam, Vahid Majdinasab, Foutse Khomh, and Michel Desmarais. 2024. Effective test generation using pre-trained Large Language Models and mutation testing. *Information and Software Technology* 171 (04 2024), 107468. https://doi.org/10.1016/j.infsof.2024.107468

[20] A. Fergusson. 2016. Designing online experiments using Google Forms + Random Redirect Tool. https://teaching.statistics-is-awesome.org/designing-online-experiments-using-google-forms-random-redirect-tool. Accessed: 2025-06-15.

[21] Jannik Fischbach, Julian Frattini, Andreas Vogelsang, Daniel Mendez, Michael Unterkalmsteiner, Andreas Wehrle, Pablo Restrepo Henao, Parisa Yousefi, Tedi Juricic, Jeannette Radduenz, and Carsten Wiecher. 2023. Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study. *Journal of Systems and Software* 197 (2023), 111549. https://doi.org/10.1016/j.jss.2022.111549

[22] Gordon Fraser and Andrea Arcuri. 2011. EvoSuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (Szeged, Hungary) *(ESEC/FSE '11)*. Association for Computing Machinery, New York, NY, USA, 416–419. https://doi.org/10.1145/2025113.2025179

[23] Kenish Rajesh Halani, Kavita, and Rahul Saxena. 2021. Critical Analysis of Manual Versus Automation Testing. In *2021 International Conference on Computational Performance Evaluation (ComPE)*. 132–135. https://doi.org/10.1109/ComPE53109.2021.9752388

[24] Benedikt Hauptmann, Maximilian Junker, Sebastian Eder, Lars Heinemann, Rudolf Vaas, and Peter Braun. 2013. Hunting for smells in natural language tests. In *ICSE 2013*. 1217–1220.

[25] Kseniia Horina and Karatanov Oleksandr. 2023. Advantages of Automated Testing of Medical Applications and Information Systems Using Gherkin and Behavior-Driven Development. In *Conference on Integrated Computer Technologies in Mechanical Engineering–Synergetic Engineering*. Springer, 379–391.

[26] Herb Krasner. 2021. The cost of poor software quality in the US: A 2020 report. *Proc. Consortium Inf. Softw. QualityTM (CISQTM)* 2 (2021), 3.

[27] Hareton KN Leung, Li Liao, and Yuzhong Qu. 2007. Automated support of software quality improvement. *International Journal of Quality & Reliability Management* 24, 3 (2007), 230–243.

[28] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013/

[29] Clementine Nebut, Franck Fleurey, Yves Le Traon, and Jean-Marc Jézéquel. 2006. Automatic Test Generation: A Use Case Driven Approach. *Software Engineering, IEEE Transactions on* 32 (04 2006), 140– 155. https://doi.org/10.1109/TSE.2006.22

[30] Wendkuuni C. Ouedraogo, Kader Kabore, Haoye Tian, Yewei Song, Anil Koyuncu, Jacques Klein, David Lo, and Tegawende F. Bissyande. 2024. LLMs and Prompting for Unit Test Generation: A Large-Scale Evaluation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 2464–2465. https://doi.org/10.1145/3691620.3695330

[31] Myron Peixoto, Davy Baía, Nathalia Nascimento, Paulo Alencar, Baldoino Fonseca, and Márcio Ribeiro. 2025. On the Effectiveness of LLMs for Manual Test Verifications. In *2025 IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest)*. 45–52. https://doi.org/10.1109/DeepTest66595.2025.00012

[32] Kostadin Rajkovic and Eduard Paul Enoiu. 2022. *NALABS: Detecting Bad Smells in Natural Language Requirements and Test Specifications*. Technical Report. Mälardalen Real-Time Research Centre, Mälardalen University. http://www.es.mdu.se/publications/6382-

[33] Prerana Pradeepkumar Rane. 2017. *Automatic Generation of Test Cases for Agile using Natural Language Processing*. Master's thesis. Virginia Polytechnic Institute and State University, Blacksburg, VA. http://hdl.handle.net/10919/76680 Advisors: Thomas L. Martin, A. Lynn Abbott, Steven R. Harrison.

[34] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[35] Gerard Salton, Edward A. Fox, and Harry Wu. 1983. Extended Boolean information retrieval. *Commun. ACM* 26, 11 (Nov. 1983), 1022–1036. https://doi.org/10.1145/182.358466

[36] Elvys Soares, Manoel Aranda, Naelson Oliveira, Márcio Ribeiro, Rohit Gheyi, Emerson Souza, Ivan Machado, André Santos, Baldoino Fonseca, and Rodrigo Bonifácio. 2023. Manual Tests Do Smell! Cataloging and Identifying Natural Language Test Smells. In *ESEM 2023*. 1–11.

[37] Elvys Soares, Márcio Ribeiro, and André Santos. 2024. A Multimethod Study of Test Smells: Cataloging Removal and New Types. In *Proceedings of the XXIII Brazilian Symposium on Software Quality (SBQS '24)*. Association for Computing Machinery, New York, NY, USA, 676–686. https://doi.org/10.1145/3701625.3701699

[38] Manoel Terceiro, Antônio Wagner, Eduardo Barros, Márcio Ribeiro, Baldoino Fonseca, Alessandro Garcia, Fabio Palomba, and Ivan Machado. 2025. Image2Test: Using ChatGPT to Build Manual Tests from Screenshots. (8 2025). https://doi.org/10.6084/m9.figshare.29482043.v1

[39] Ubuntu. 2024. *Ubuntu Manual Tests*. https://launchpad.net/ubuntu-manual-tests

[40] Ravi Prakash Verma and Md Rizwan Beg. 2013. Generation of test cases from software requirements using natural language processing. In *2013 6th International Conference on Emerging Trends in Engineering and Technology*. IEEE, 140–147.

[41] Chunhui Wang, Fabrizio Pastore, Arda Goknil, and Lionel C Briand. 2020. Automatic generation of acceptance test cases from use case specifications: an nlp-based approach. *IEEE Transactions on Software Engineering* (2020).

[42] Yuqing Wang, Mika V Mäntylä, Zihao Liu, and Jouni Markkula. 2022. Test automation maturity improves product quality—Quantitative study of open source projects using continuous integration. *Journal of Systems and Software* 188 (2022).

[43] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382* (2023).

[44] Dianxiang Xu, Weifeng Xu, Michael Kent, Lijo Thomas, and Linzhang Wang. 2014. An automated test generation technique for software quality assurance. *IEEE transactions on reliability* 64, 1 (2014), 247–268.

[45] Tao Yue, Shaukat Ali, and Man Zhang. 2015. RTCM: a natural language based, automated, and practical test case generation framework. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis* (Baltimore, MD, USA) *(ISSTA 2015)*. Association for Computing Machinery, New York, NY, USA, 397–408. https://doi.org/10.1145/2771783.2771799

[46] Yixue Zhao, Saghar Talebipour, Kesina Baral, Hyojae Park, Leon Yee, Safwat Ali Khan, Yuriy Brun, Nenad Medvidović, and Kevin Moran. 2022. Avgust: automating usage-based test generation from videos of app executions. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 421–433. https://doi.org/10.1145/3540250.3549134