Automating Test Design Using LLM - Results from an Empirical Study on the Public Sector

Artur Raffael Cavalcanti[a], *Luan* Accioly[b], *George* Valença[c], *Sidney* C. Nogueira[d*], *Ana* Carolina Morais[e], *Antônio* Oliveira[f], *Sérgio* Gomes[g],

[a]Computer Science Department, Federal Rural University of Pernambuco, UFRPE, Brazil, artur.raffael@ufrpe.br, 0009-0005-9079-766X.

[b]Computer Science Department, Federal Rural University of Pernambuco, UFRPE, Brazil, luan.accioly@ufrpe.br

[c]Computer Science Department, Federal Rural University of Pernambuco, UFRPE, Brazil, george.valenca@ufrpe.br, 0000-0001-9375-5354.

[d]Computer Science Department, Federal Rural University of Pernambuco, UFRPE, Brazil, sidney.nogueira@ufrpe.br, 0000-0002-8817-5029.

[e]Tribunal de Contas do Estado de Pernambuco, TCE-PE, Brazil, anacarolina@tcepe.tc.br

[f]Tribunal de Contas do Estado de Pernambuco, TCE-PE, Brazil, antoniolira@tcepe.tc.br

[g]Tribunal de Contas do Estado de Pernambuco, TCE-PE, Brazil, sergiogomes@tcepe.tc.br

Submitted: 31 January 2025, Revised: 26 March 2025, Accepted: 21 April 2025, Published: 23 May 2025

Abstract. An efficient test process can detect failures earlier in software development, contributing to the quality of software produced by governmental entities and bringing the potential to improve government service delivery. Due to the high cost and the reduced resources available, the automation of the test activities plays a strategic role in improving testing efficiency. Designing test cases from user stories is a common approach to assessing the quality of a system under testing. This paper reports on the research, implementation, and evaluation of a tool that automatically generates system test designs from user stories with the support of Generative Pre-trained Transformer 4 (GPT-4) in the context of a public sector organization. The tool has been conceived to match the needs and particularities of the organization's test process. Such a tool reads user stories from the Redmine tool, interacts with GPT-4 using a prompt that outputs test cases, and stores the automatically designed tests in the Squash TM test management tool. Organization test analysts stated that the tool produces good quality tests and reduces the effort to create tests. As a consequence, analysts can put more energy into other activities related to testing. Moreover, comparing the tests designed manually by test analysts with the tests designed by the tool shows that both have the same functional coverage. The paper discusses the impacts of the approach in the process, limitations, and related and future work.

Keywords. automated test design, LLM, user story, system test cases
Research paper, DOI: https://doi.org/10.59490/dgo.2025.1025

1. Introduction

Software solutions are an essential part of the services provided by public sector organizations. In such a context, software testing (Myers & Sandler, 2004) is necessary to assess the quality of software products and reduce operational costs due to software failures. Failures detected later cost much more to be corrected than

those detected earlier (Kushwaha & Misra, 2008). Improving the quality of software produced by governmental entities has the potential to improve government service delivery. The cost of software testing may be higher than 50% of the total cost of software development (Myers & Sandler, 2004). Due to such high costs and reduced testing resources, the automation of test activities plays a strategic role in improving testing efficiency. Surveys (Garousi & Varma, 2010; Ng et al., 2004) conducted in the public sector show that automation of test activities is a demand to make the testing process more efficient. Moreover, testing methodologies (Nomura et al., 2013; Sivaji et al., 2011) and quality models (Kushwaha & Misra, 2008; S Al- Tarawneh & Al-Tarawneh, 2022) have been proposed to improve the effectiveness of testing in the public sector. Propositions recommend the use of automation as a tool to improve the test process.

Test cases are vital in testing (Myers & Sandler, 2004); each test is an experiment that defines the inputs provided during test execution and the expected outputs produced by the System Under Testing (SUT). The demand for tests increases proportionally to the size of the application. Test design is a critical activity in software testing that focuses on creating test cases; it is not trivial and demands specialized skills. It is present in almost all testing levels and testing types (Sommerville, 2007). A well-designed test case increases the chance of finding important issues in the system's behavior. The automation of test design is a relevant approach to improving test effectiveness. System testing is a testing level where the test cases mimic the final test analyst interaction with the SUT. At such a level, tests are created based on the system's specifications (black box approach). Thus, system testing design requires interpreting existing documentation that functions as the test basis. Notably, user stories (Sommerville, 2007) are a fundamental component of agile software development; they are lightweight for capturing and managing requirements. User stories are a standard test basis for test design in agile processes.

The context of the current research is the software development process of a public sector organization that works with auditing and accountability, namely Tribunal de Contas do Estado de Pernambuco (TCE-PE)[1]. Such an organization produces web systems that support its services to the population. In the TCE's software development process, test analysts receive user stories as the main input for the design of system-level test cases that are executed to validate the correct behavior of the produced software. Furthermore, the organization lacks sufficient resources to design and execute tests for its systems. The demand for testing is greater than the number of existing human resources (Lins et al., 2023). The partial automation of test design would reduce the effort in the design so that test analysts would put more effort into running the tests and detecting defects to be corrected as early as possible.

Previous works have explored test generation from user stories for producing system tests (Chinnaswamy et al., 2024; Granda et al., 2021; Mollah & van den Bos, 2023; Pallavi Pandit, 2015). However, the existing solutions differ in the format for the user stories, the format and the purpose of the produced tests, and the approach used for test generation. Moreover, related work has not been motivated or evaluated in the context of a public organization, as in the current work.

The current paper addresses three research questions.

R1 How to build a tool that reads user stories and automatically designs system test cases?
R2 How do the test cases designed by the tool compare to the tests designed by the test analysts?
R3 What are the test analysts' perceptions of the tool and the automatically designed test cases?

To answer the first question, we created a prototype for a tool that uses an LLM to automate the design of system tests for manual execution, considering the tools used for testing the software produced by the public organization of this study. The proposed prototype interacts with ("GPT-4", 2025) to obtain system tests by inputting user stories. The interaction uses a prompt following the few-shot technique crafted with examples of user stories and the respective test cases in the desired format; then, original test cases are yielded by the LLM based on user stories that have not been provided as examples. Such a tool has been implemented in Python and provides a command line interface that allows test analysts to generate system test designs from user stories that are read from the Redmine tool, and the automatically designed tests are stored in the Squash TM (*Squash TM*, 2025) test management tool. To answer the second question, we compared the tool-designed tests with the existing ones with respect to the number of tests, steps, and functional coverage. It could be verified that the functional coverage of the tests designed by the tool is similar to that designed by the analysts. Moreover, the number of tests and steps is very similar to those created by the analysts. To answer

[1] https://www.tcepe.tc.br/internet/

the third question, we created an anonymous questionnaire to collect the perceptions of the test analysts and also conducted a meeting to collect complementary feedback for the tool. After using the tool to support test design in their habitual process and environment for about a month, test analysts stated the tool reduced the overall effort for test design by 60% significantly improving efficiency. It also simplifies the use of Squash TM by automating mechanical tasks, which are often a significant obstacle for analysts, as noted in their feedback. The feedback also included points for improving the tool.

The remainder of this paper is organized as follows. The next section introduces the organizational context of this work and the main concepts that outline this work. Section 3 overviews the steps of the research method. It describes our eight-step process followed in this research. Subsequently, Section 4 details the tool's architecture and the tool workflow from the point of view of the test analyst. Afterward, Section 5 presents details on the evaluation and key findings supported by collected data. Then, Section 6 discusses the research questions in light of the evaluation results and their implications. Finally, Section 7 summarizes contributions, presents related work, and discusses future work.

2. Context and conceptual background

This section overviews the relevant portion of the testing process of the organization in the public sector that contextualizes this research. In such a process, the project is organized as sprints, as shown in (Augustine, 2005). On each sprint, requirement analysts create user stories and store them in Redmine. Then, developers and test analysts use the stories as the basis for coding the application and for designing test cases using Squash TM, respectively. Moreover, Squash TM is used by test analysts to create an execution plan with a list of tests to be executed. After the developers release a version for testing, the test analysts execute the tests following the execution plan and keep a record of the status of each individual test. Failed tests are reported to the developers to fix the software. Failing tests are re-executed after the fixes. After releasing a software version that passes the tests, the product owner validates the new version, which can lead to new bugs and fixes. After successive interactions, the software has all its features implemented and can be deployed for the end users of the public organization. In what follows, we show the format for user stories and test cases and the tools used to manipulate these artifacts.

This paper does not exhibit internal artifacts of the TCE-PE software development process for confidentiality reasons. The user story and test cases presented in this section follow the format of the real artifacts; however, they do not relate to a real system.

2.1. User stories and Redmine

A user story (US) facilitates communication between technical teams and end test analysts, ensuring that the development process remains user-centric and aligned with stakeholder needs. The Redmine tool (*Redmine*, 2006) is an open-source project management and issue-tracking tool that supports agile workflows and efficiently organizes sprints and user stories. Also in Redmine, it is possible to define user stories and link them to sprints, creating an organized flow.

Figure 1 shows a sample US, a simplified version of the format used in the public organization that contains the relevant fields for test case design. The US in Figure 1 specifies the behavior for the system login page. Each US is identified by a unique ID, in the sample, the ID equals 00001. This information is relevant for relating tests to the respective source US. The US title (displayed in the first line of the figure offers a quick reference to the purpose of the US, helping to identify the area of functionality the tests will cover. The title is preceded by the `PROJ` prefix, which is a sample acronym for the project that the US belongs to. The description provides context about the user's goal and the functionality's purpose. The scenarios field outlines the flows of the SUT behavior. In the sample US, there are two scenarios, the first scenario evaluates the system's ability to authenticate a registered user with valid credentials, ensuring successful login and redirection to the homepage. The second scenario tests the system's response to invalid credentials, verifying that an appropriate error message is displayed, preventing unauthorized access while providing clear user feedback. Acceptance criteria specify the expected outcomes and requirements that must be met; in the sample, there are four. A test analyst must design test cases to verify the condition described in each criterion of a US.

User Story #00001

PROJ - *As a registered user, I want to be able to log in to the platform to access my profile and exclusive content.*

Description
This feature enables registered users to securely log into the platform, providing access to personalized content and their profiles. If users enter incorrect credentials, a clear error message is displayed, and if they forget their password, a reset option is available.

Scenarios

1. **Successful login with valid credentials:**
   - Given a registered user with valid credentials, when the user enters their correct username and password and clicks the login button, then the user is successfully authenticated and redirected to the platform's homepage, where exclusive content is accessible.
2. **Login with invalid credentials:**
   - Given a registered user attempts to log into the platform, when the user enters an incorrect username or password and clicks the login button, then a clear error message is displayed, indicating that the login information is incorrect.

Acceptance Criteria
a) As a registered user, I want secure access to the platform to view my profile and personalized content.
b) I want to enter my username and password to log into my account.
c) After successfully logging in, I want to be redirected to the platform's homepage, where I can begin exploring the exclusive content available to me.
d) If I enter invalid credentials, I expect to receive a clear error message indicating that my login information is incorrect.

Fig. 1 – User Story

2.2. Test cases and Squash TM

A classical format for a test case design for system testing (Sommerville, 2007) specifies preconditions (pre-requisites) and execution steps. Preconditions specify the necessary conditions (setup actions) to start the test. Each step contains the inputs provided to the SUT and the expected outcomes. The execution of a test case is an experiment to verify that the SUT meets its specification. If some step fails during the execution, we say the test fails and the execution evidences a defect in the SUT. A defect is a deviation from the expected behavior described in the US. If no step fails, the SUT passes the test.

Key requisites for a good test case include clarity, repeatability, and traceability to user stories. A set of test cases covers the functionality of a set of user stories if there is at least one test case to verify the scenarios and criteria specified in the stories. Figures 2 display tests in the interface of Squash TM that cover the US shown in Figure 1.

Figure 2 presents the key fields of tests in Squash TM, with non-essential fields omitted. At the top of the figure, the title field is highlighted in green. The title should clearly identify a test. Each test has a unique ID that appears in the Figure 2 as `CT-PROJ-2`. This tag is the composition of the prefix `CT` with the project acronym (in the sample `PROJ`) and a number. The test status and prerequisites are displayed below the title; they are followed by the actions (inputs) and the expected results. The test analyst is responsible for verifying that these actions comprehensively cover the scenarios and meet the acceptance criteria established in the related US.

In Squash TM, test cases can be organized in folders. A helpful structure is to create a folder of test cases for each Sprint. There is also the Requirement field, which includes a reference to the user stories that are used as the basis for writing the test case. This field allows linking tests to user stories, ensuring traceability.

3. Overview of the research method

This research seeks to answer the three research questions listed in the introduction. For this purpose, it followed eight steps.

1. Selection of existing user stories and test cases to be used as references.

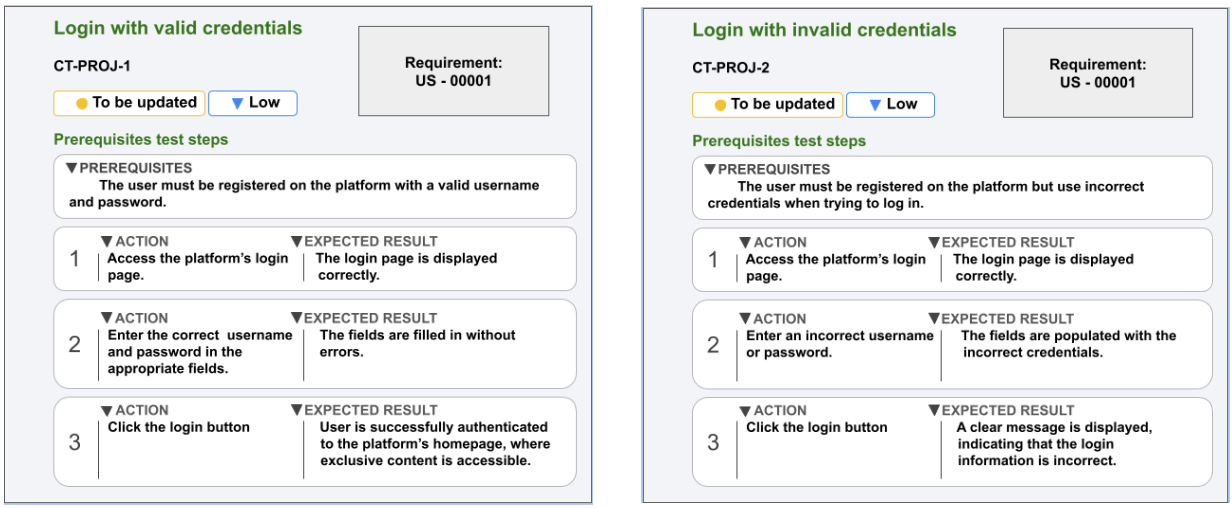

Fig. 2 – Tests created by the US2Test Tool for US 00001

2. Implementing the tool US2Test to automatically design tests from user stories using LLM.
3. Comparing the test cases designed by the tool to the existing test cases.
4. Sharing comparison results with test analysts to collect feedback about the automatically designed tests.
5. Distributing US2Test to the test analysts.
6. Collecting feedback from test analysts using an anonymous survey.
7. Scheduling a meeting with the test analysts to collect direct feedback.
8. Analysis of the results.

We explain the above step. First, the organization's quality assurance team provided reference artifacts consisting of well-designed test cases created to cover a predefined set of user stories. The reference artifacts follow the organization patterns and guidelines for having good-quality user stories and test cases. Such artifacts have been used in previous versions of software developed by the organization. The second step was to develop the US2Test tool that uses LLM to generate tests from the US. The implemented tool interacts with GPT-4 using OpenAI API to design test cases. The third step focused on using the tool to design tests automatically and compared this suite to the test suite designed by test analysts (provided in the first step).

The fourth step consisted of sharing the comparison results with analysts to gather their impressions before they start using the tool. Based on the feedback received, we could enhance the tool and distribute the binaries to the test analysts (fifth step) for their daily use. In this way, they could design tests using the tool by themselves. The sixth and seventh steps involved collecting further feedback after the analysts had actively used the tool. Initially, feedback was gathered anonymously, and the forms were distributed approximately one month after the test analysts gained access to the tool. About two months after accessing the tool, a meeting was scheduled to collect more direct and detailed insights. Finally, we analyzed all the data and started writing this paper.

4. Tool for automatic test design using LLM

The automatic test generation approach introduced in this section is tailored for use in the test process of TCE-PE (presented in Section 2). Therefore, the development follows the particularities of the organization's testing process. Figure 3 illustrates the proposed test generation approach implemented by a tool called US2Test. In the figure left-hand side (in cyan) are the primary inputs for test generation: examples of user stories stored in Redmine (`Example US (Redmine)`) with their respective test case (`Example US (Redmine)`) and the user stories that we want to generate new tests (`Input user stories (Redmine)`). Examples are retrieved from Redmine and Squash TM using the APIs for such tools. User stories and tests follow the format introduced in Section 2 (refer to Figures 1 and 2). The inputs are translated into JavaScript Object Notation (JSON) format (blue color) and processed by a Python script that creates a prompt that is the input for the GPT-4 API. The GPT-4 model processes the prompt and yields a set of test cases encoded in JSON format for the desired user stories. The set of test cases is then pushed to the appropriate location in the API of the Squash TM tool.

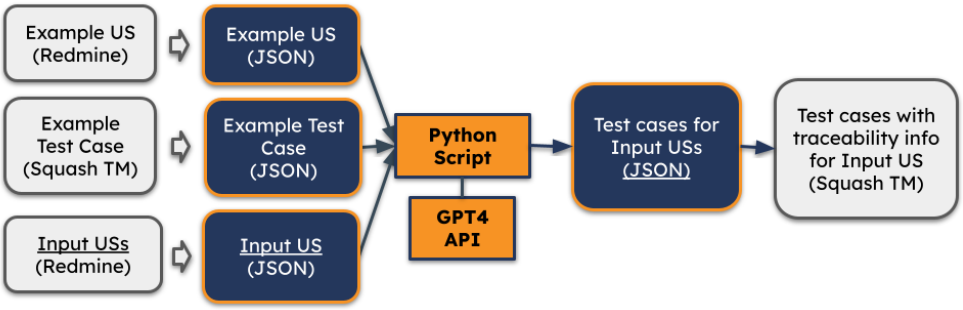

Fig. 3 – Workflow for the US2Test tool

The tool performs (Bahrami et al., 2023) to guarantee that the automatically designed test cases follow the classical test case format, which is one of the formats accepted by the Squash TM tool. Moreover, by providing examples of user stories and test cases, we help the model understand the match between the user stories and test cases so that tests designed for the input user stories follow a predefined style for the tests.

Listing 1 illustrates the prompt used in our approach to generate tests. This prompt follows the RTF (Role Task Format) format, which instructs the prompt on its role, the task it will perform, the rules it will follow, the format for the inputs, and the format for the expected response. The prompt specifies rules to maintain consistency, such as ensuring that each test step has a corresponding expected result. Additionally, each test case must have preconditions, and only the JSON with the test cases should be returned without extra information. The placeholders `<EXAMPLE USER STORIES>`, `<EXAMPLE TEST CASES>`, and `<USER STORIES>` represent, respectively, the sample user stories, test cases for the sample user stories, and the stories we want to generate new tests. Such placeholders are replaced by JSON objects representing content retrieved from Redmine and Squash TM. Likewise, the prompt instructs the model to use specific black-box testing techniques, such as Equivalence Class and Boundary Analysis (Myers & Sandler, 2004). These techniques are employed to design tests that verify significant cases and edge cases for the scenarios described in the input user stories.

Listing 1 – Few-shot Prompt

```
You will act as a test case creator. You will receive a user story in JSON format and should
    return enough test cases to cover the user story, considering the use of black-box techniques
     such as equivalence class and boundary values. The result must be returned in JSON format (
    RFC 8259). Some rules: For each test step, there must be an expected result; The number of
    steps and expected results must be the same, so that an expected result is linked to a step;
    Steps that consist of verifying something must be in the expected result field; Each test
    case must contain preconditions; You should return only the JSON with the Automatically
    designed test cases, without any additional information.
Here is an example of a user story: <EXAMPLE USER STORIES>
The test cases for this user story are: <EXAMPLE TEST CASES>
Generate test cases for the following use stories: <USER STORIES>
```

The code snippets in Listings 2 and 3 are JSON objects that represent the US and the test cases introduced in Section 2.1. The snippets are abbreviated for conciseness. The placeholders `OTHER CRITERIA`, `OTHER SCENARIO`, and `SECOND TEST HERE` indicate the points that have been omitted. The omitted criteria require successful login, redirection upon authentication, and access denial for incorrect credentials. The concealed scenario specifically examines failed login attempts, while the hidden test verifies that access is denied following this scenario.

Listing 2 – Sample US enconded in JSON

```
{
  "title": "As a registered user, I want to be able to log in to the platform to access my
      profile and exclusive content.",
  "description": "This feature enables registered users to securely log into the platform,
      providing access to personalized content and their profiles. If users enter incorrect
      credentials, a clear error message is displayed, and if they forget their password, a reset
       option is available.",
```

```
  "acceptance_criteria": [
    "As a registered user, I want secure access to the platform to view my profile and
        personalized content.",
    < ~OTHER CRITERIA~ >
  ],
  "scenarios": [ {
     "title": "Successful login with valid credentials",
     "Given a registered user with valid credentials, when the user enters their correct username
          and password and clicks the login button, then the user is successfully authenticated
         and redirected to the platform's homepage, where exclusive content is accessible."
    },
    { ~OTHER SCENARIO~ } ]
}
```

Listing 3 – Sample Test Case enconded in JSON

```
"US00001": {
  "test_cases": [ {
     "title": "Login with valid credentials",
     "pre_conditions": ["The user must be registered on the platform with a valid username and
         password."],
     "steps": [
       "Access the platform's login page.",
       "Enter the correct username and password in the appropriate fields.",
       "Click the login button."
     ],
     "expected_result": [
        "The login page is displayed correctly.",
        "The fields are filled without errors.",
        "User is successfully authenticated and redirected to the platform's home page where the
            exclusive content is accessible."
        ]
     }, { ~SECOND TEST HERE~ } ] }
```

The tool stores generated tests in a particular folder of Squash TM as exhibited in Figure 4. The tests in this figure represent the tests depicted in Figure 2.

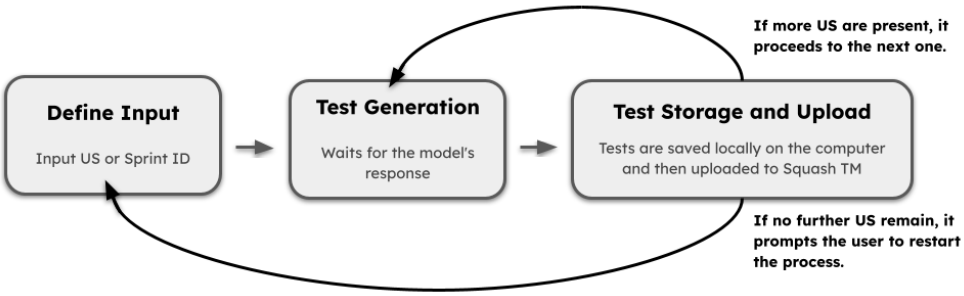

Fig. 4 – Test analyst workflow

We explain the steps to use the tool. The tool´s command line interface follows a structured process, as outlined in Figure 4. Before starting to use the tool, the test analyst needs to include the credentials for Redmine and Squash TM in a configuration file. As the first step, the test analyst needs to define the inputs for the test design. It can choose between generating tests from individual user stories or sprints. Using the first option, the test analyst can manually input one or more IDs for user stories. In the second option, the test analyst can provide a sprint ID, and the tool will yield tests for all user stories associated with the sprint.

After the test analyst defines the inputs, the tool initiates the test generation process for each US. This step is typically the most time-consuming, as it relies on the LLM. Errors can occur during the generation, even if the inputs respect all formatting rules. The model generates malformed outputs, so the tool needs to verify

whether the tests are syntactically valid. If errors are detected, the tool automatically initiates a new interaction with the model to redesign the tests, with up to five attempts.

Once the test generation has finished successfully, the test cases are locally stored on the computer and then uploaded to Squash TM. For the upload, the test analyst should provide the target folder in Squash TM (mandatory) and, optionally, the project acronym (default value is PROJ if not specified) and the test number (default value is one if not provided).

Finally, after processing one US, the tool checks if there are more user stories to process. If so, it proceeds to the next one and starts a new iteration for preparing new tests. Otherwise, it restarts the process by asking the test analyst to provide new inputs.

5. Evaluation

This section presents the execution of the evaluation as well as its results in the public sector organization, which is the context of this work. It further details the third to the seventh steps of the research method presented in Section 3.

5.1. Evaluating the tests designed by the tool

A very relevant part of the evaluation is to compare the test cases designed by the LLM to the existing test cases and to collect feedback for the automatically designed tests from the test analysts. This section reports on both parts.

The comparison used as a reference the set of seven US provided by the quality assurance team (see Section 3). Such a set was split into two disjoint subsets: a set with three user stories (and its respective test cases) to be used as examples for the prompt (see Section 4), and the second set containing four user stories used as input to create new tests using the tool US2Test. The tests designed by the tool have been compared to those designed by the test analysts.

For each reference US, we collected two measures and one metric to compare tests produced by the tool and those created by the analysts.

Number of Tests Records the total number of tests designed for a particular US. The number of tests indicates the number of separate test cases developed to verify the user stories. It reflects the test suite's depth to cover different scenarios through distinct tests. A very small number of tests would increase the risks of potential defects not being detected (Myers & Sandler, 2004). On the other hand, many tests are not welcome since they cannot be feasible for test execution due to resource availability.

Number of Steps This measure represents the sum of the number of steps of the tests designed for a particular US. The number of steps in each test case provides insight into the level of detail and granularity of the actions required to carry out each test. This measures the complexity and thoroughness of each test case, reflecting how procedural actions are broken down.

Test suite coverage Stands for the coverage of a test suite. Let TS be the sum of the number of scenarios and acceptance criteria for a particular US, and CS represent the number of test scenarios covered by a test suite created for the same US. This metric is obtained by the expression $CS/TS * 100$ that records the percentage of scenarios and criteria of a given US that are addressed by at least one test in the test suite. This metric is essential to evaluate the quality of a test suite. Complete (100%) or near-complete coverage ensures that the test cases address all functional paths and requirements, minimizing the risk of missed defects.

Table 1 shows the measures for the tests designed by the analysts (MDT — Manually Designed Tests) and for those designed using the tool (ADT — Automatically Designed Tests). The table consists of four columns: US, Manually designed tests, Automatically designed tests, and Difference. The US column represents the ID of each US used in the evaluation. The columns Manually designed tests and Automatically designed tests refer to tests created by analysts and the tool, and the Difference column refers to the difference in the metrics between MDT and ADT. The last three columns are further divided into two sub-columns: Test and Steps. These sub-columns refer to the total number of tests and steps.

US	Manually designed tests		Automatically designed test		Difference	
	Tests	Steps	Tests	Steps	Tests	Steps
1	2	4	2	6	0	2
2	2	4	4	16	2	12
3	2	2	2	7	0	5
4	2	2	2	6	0	4
Sum	8	12	10	35	2	23

Tab. 1 – Tests comparison table

Table 1 shows differences in the number of tests and steps in MDT and ADT. The difference shows that while the test quantity typically remains consistent between the two methods, the number of steps is often higher in the ADT. Despite this difference, all tests and steps in ADT are related to some tests and steps in MDT. We report the main differences noted between the number of test steps in ADT and MDT.

1. Tests in MDT benefit from the analyst's knowledge of the system's user interface (UI), enabling the definition of optimized sequences of actions to execute. In contrast, tests in ADT are solely based on the US content since the prompt is not fed with UI information; thus, designed tests occasionally include intermediate navigation steps the LLM deems necessary to reach the appropriate screen.
2. Tests in ADT systematically include a final step to confirm the save operation to persist changes. While save actions are left implicit in the analyst-created tests.

Similar observations can explain the difference in the number of tests in ADT and MDT for US 2. The first suite contains four tests, and the second two. Putting the tests of both suites side by side, we could observe that the four tests in the first suite, if merged, would yield the two tests in MDT. Such tests could be merged since they can be executed on the same application page. The model does not have information about the UI, and created independent tests that could form a unique test to be executed on the same application page.

These differences highlight how the LLM prioritizes a detailed and explicit description of each action, even at the cost of increasing the number of steps. This approach contrasts with the optimized methodology adopted by analysts, who leverage system knowledge to streamline the test execution process.

The coverage metric is not included in Table 1 since MDT and ADT achieved 100% coverage for each US listed in the table. The collection of coverage was conducted by one of the authors of this paper, who looked for each criterion and scenario in each US and looked for the tests in the suite to find if some steps were referring to the criterion and scenario. This leads to the calculation of 100% coverage of the user stories by ADT and MDT. This means both test suites have completely covered the input user stories used for designing the tests. Consequently, both test suites demonstrate adherence to the acceptance criteria and defined scenarios for each US.

In the meeting to collect feedback with the test analysts (refer to step four in Section 3), despite the differences noted in the number of tests and steps, the analysts considered the tests in ADT as "good tests", since they are coherent to user stories, are easy to follow and have a clear language. Test analysts reported being motivated to use the tool in their daily activities to better assess its functionality and the quality of the generated test cases.

Despite not interacting directly with the system's interface, the approach proved highly effective, accurately executing and validating actions, including screen interactions.

5.2. Collecting survey data

As part of our research method, an evaluation was conducted through an anonymous survey answered by analysts who used the tool US2Test. The survey included questions to assess the tool's usability, effectiveness, and impact on their workflow. Additionally, it has questions to collect the perception of the test cases designed by the tool. The questions in the survey are:

Q1 Does the US2Test tool have any impact on the efficiency of your daily work?

Q2 How much do you consider the tool facilitates the manual test creation process?
Q3 How much do you consider the tool facilitates the process of sending to Squash TM?
Q4 List points you liked and disliked about the tool.
Q5 Are the automatically designed tests clear and easy to understand?
Q6 The automatically designed tests adequately cover the user story scenarios?
Q7 The automatically designed tests can help identify potential errors or issues?
Q8 What aspects of the automatically designed tests could be improved to better meet your needs?

The first four questions assess the tool's effectiveness, while the last four questions evaluate the quality of the automatically designed tests. Two questions are open-ended (questions four and eight), and the others are multiple-choice. The multiple-choice questions were designed following the (*Likert scale*, 2006), a popular method for measuring attitudes or opinions. On this scale, respondents rate each statement on a scale from one to five, where one means they completely disagree or find it unhelpful, and five means they agree entirely or find it highly helpful. Number three means a neutral opinion.

The survey was distributed by email to a group of four test analysts: this is the total number of test analysts available to use US2Test. Represents the analysts of a particular project inside the public organization. Moreover, filling out the survey was not mandatory. The answers for the multiple-choice questions of a group of three test analysts are recorded in Table 2. The responses to the open-ended questions are exhibited in the Appendix.

Questions	Test analyst 1	Test analyst 2	Test analyst 3
Q1	4	3	5
Q2	4	3	5
Q3	5	3	5
Q5	4	4	5
Q6	4	4	5
Q7	4	4	5

Tab. 2 – Test analysts' opinion about the tool and its tests

Each column in the table represents the opinion of a different test analyst, with responses based on a Likert scale. Analyzing the responses to the first three questions, we observed that most of the feedback was positive or neutral. One test analyst was neutral and totally agreed that the tool offered significant assistance. Therefore, concerning the tool's effectiveness, we can observe that it is helpful but not a complete paradigm shift in daily routines. In the subsequent three questions, two-thirds of the respondents indicate that the tests are clear and beneficial, while one-third find them exceptionally clear.

Reading the responses to Q4 and Q8 in the Appendix, we identify different test analyst perspectives. Test analysts state the tool is useful, and at the same time, there is a consensus on the need for improvements, especially regarding fixing errors during the automatic creation of tests. For the eighth question, the feedback highlights that the effectiveness of test creation is closely tied to the quality of user stories. There is room for improvement in the tools' ability to cover all scenarios in more complex user stories.

Data in Table 2 indicates significant potential for improvement in the tool. The automatically designed tests are currently satisfactory, adhering to predefined scenarios, aiding in error identification, and being clear and comprehensible.

5.3. Feedback meeting

To complement the anonymous feedback, we conducted a meeting to gather additional feedback from the test analysts. This meeting was about one month after the responses to the survey had been collected. Thus, reflects a possible more mature opinion about the tool's impacts. During the meeting, it was clearly stated that the tool prototype had demonstrated productivity gains and is frequently used by at least two team members. After that, one analyst reported that the tool reduces up to 60% of the effort in creating tests. Although the tests designed by the tool require further adjustments to be ready for test execution, it was agreed that the tests are a quality starting point for creating tests with quality.

Addressing environmental issues that prevent some analysts from using the tool (for instance, an anti-virus that blocks the tool from working properly) and fixing defects was identified as a priority. Furthermore, during the meeting, the analysts emphasized the desire for the tool to create tests for all the sprints in a given project. Currently, the tool generates tests for all user stories in a sprint but does not generate tests for all sprints in a project.

Finally, implementing a simple graphical user interface was proposed to enhance the overall test analyst experience by reducing repetitive data entry. The meeting concluded with a consensus that the specific requirements for this interface will be discussed with the tool test analysts later.

6. Discussion

Considering the evaluation presented in Section 5, we can go back to discuss the research questions introduced in Section 1.

6.1. [R1] How to build a tool that reads user stories and automatically designs system test cases?

The response to this answer is detailed in Section 4. Thanks to the capabilities of LLMs in natural language processing, the automation of tasks that involve input and output of textual content is facilitated by the current LLMs, such as GPT-4, which has been used in this research. We made a relatively small effort to set a prompt and refine it until we got to the version presented in this study. A greater amount of time was dedicated to the automation of the approach presented in Figure 3, which requires integration with Redmine and Squash TM and the construction of a simple command-line tool. From this experience, we believe LLMs can facilitate the construction of different tools to support software testing activities.

6.2. [R2] How do the test cases designed by the tool compare to the tests designed by the test analysts?

The results exhibited in Section 5.1 bring interesting points for discussion. It could be observed that all ADT tests are related to some MDT tests and that ADT has complete coverage in the same way as MDT does. Despite that, there are differences in the number of tests and steps in both suites. The test cases automatically designed by the tool (ADT) produce a larger number of test cases and steps compared to the tests designed by the test analysts (MDT). As reported in Section 5.1, some tests in ADT, if merged, would bring similar tests in MDT. The increase in the number of steps is justified by the fact that the tests in ADT have detailed and explicit descriptions that add extra steps for the tests. While tests in MDT are more concise since they leave some points implicit. Even considering the differences, test analysts considered tests in ADT to be acceptable since they are coherent with user stories, easy to follow, and have a clear language.

A point for future investigation is to analyze the effect on the tests of ADT if UI information is provided as context information to the model. A hypothesis to be observed would be to observe whether the model would produce a smaller number of tests: possibly, individual tests would be put together because they can be executed on the same page of the SUT.

A positive aspect of the ADT tests that was not explored in the study is that the ADT tests do not contain grammatical or spelling errors. This is a positive aspect of the tests generated by the model that contributes to the quality of the tests. Moreover, due to the rules in the prompt and the format validation performed by the tool, designed tests consistently match the expected format for test cases, ensuring uniformity of tests.

6.3. [R3] What are the test analysts' perceptions of the tool and the automatically designed test cases?

Data presented in Sections 5.2 (survey answers) and 5.3 (feedback meeting) help in the discussion of the third research question. We have two dimensions to consider in this question: the perceptions about the impacts of adopting the tool and the perceptions about the tests produced by the tool.

Impacts of adopting the tool Looking at the survey answers, two-thirds of the test analysts agreed the tool has positive impacts on the efficiency of their activities, in the creation of tests, and in the process of publishing the tests into Squash TM. Considering there was a month between the initial contact with the tool and the

answers in the survey, it was a very short period for the respondents to formulate an opinion about the tool. An indication of this is the answer of Test Analyst 3 to the eighth question in the survey (refer to Appendix). This respondent explicitly states that he has just started using the tool. Conversely, by the time of the feedback meeting, the test analysts had been in contact with the tool for about two months. During the meeting, they expressed that the tool reduces up to 60% of the effort in creating tests. Also, they provide a rationale for this reduction: the tests designed by the tool are a very good starting point, but require further adjustments to be ready for test execution. This reinforces the feeling that LLMs are productivity tools rather than substitutes for human skills (Ross et al., 2023).

Perceptions about the produced tests In the last four questions on the survey, all respondents agreed that the tests designed by the tool were clear and easy to understand, they covered scenarios and acceptance criteria of the user stories, and they helped identify issues in the SUT. Regarding coverage of the tests, the respondents' perception matches with the complete coverage reached by the automatically designed test in the evaluation reported in Section 5.1. By the time test analysts answered the survey, they had the opportunity to use the tool to input user stories of different projects and reach adequate coverage for the tests. It gives a complementary indication that the tests generated by the tool have good coverage for a variety of user stories. The issue reported by test analysts when attempting to generate test cases occurred in user stories that present a specific character. Following test analyst feedback, the root cause was identified and fixed in the new version of the tool.

7. Conclusion

The paper introduces US2Test, a tool designed to automate the generation of system test cases from user stories using Large Language Models. It integrates with existing systems to suit the test process of a particular public organization. The empirical evaluation of the tool involving the organization's test analysts demonstrated that the tool effectively helps the design of test cases. A more effective test process can contribute to the quality of software produced by governmental entities and has the potential to improve government service delivery.

7.1. Related work

Previous works have considered user stories for the derivation of tests. For instance, the authors in (Pallavi Pandit, 2015) propose a systematic approach to manually design acceptance tests from the user stories. Additionally, the work (Granda et al., 2021) introduces a model-driven framework that inputs user stories, such that the acceptance criteria are described using the GWT (Given-When-That) structure, and produces automatic test scripts for desktop applications. The authors in (Mollah & van den Bos, 2023) present a step-by-step method to manually transform user stories into automatic test cases. The format for the user stories and the tests that are produced for these related works differs from ours, which accepts a less structured story and produces system tests in the format of a test management tool.

Domain-specific textual formats have also been used to produce tests. The works (Carvalho et al., 2015) and (Nogueira et al., 2019) follow a model-based testing approach (Dias Neto et al., 2007). The authors in (Carvalho et al., 2015) proposed a strategy to automatically generate test cases from Natural Language requirements (NAT2TEST - NATural language requirements to TEST cases). This approach inputs requirements in a Controlled Natural Language (CNL) that follows a well-defined grammar and vocabulary to allow the extraction of a formal model for automatic verification of requirements and the extraction of test cases. A test case is a sequence of test vectors that is the input for some external tool that formats the test for a particular context. The work (Nogueira et al., 2019) proposes a model-based approach to generate test cases for manual execution from well-structured use-case models with well-defined semantics. Use case flows are defined using a tabular format and specified as a mix of natural language and a special CNL. The produced test cases are suitable for manual execution and follow a format very similar to the format proposed by this work.

Natural language processing (NLP) Raharjana et al., 2021 has been extensively used to analyse user stories. One of the applications of NLP is to produce tests (Chinnaswamy et al., 2024; Leotta et al., 2024). More recently, large language models (LLM) have been explored to generate tests from different textual formats (Ng et al., 2004). For instance, the work (Mathur et al., 2023) proposes an approach that uses machine learning models like T5 and GPT-3 to automate the test case generation process. The input for the approach is a statement in natural language. Such a statement is not structured as a US. Moreover, the output is an unstructured test.

7.2. Threats to validity

We discuss some of the threats to the validity of this research.

Construct Validity: The proposed approach for test generation (introduced in Section 4) feeds the GPT-4 model with a fixed prompt with a fixed set of examples of user stories. The prompt structure can influence the results. Moreover, this prompt contains examples that can potentially influence test generation and limit the generalization of the results.

Internal Validity: Since analysts frequently discussed the tool before and during the evaluation, there is the possibility of an unstructured knowledge transfer between them. This informal learning process may have biased how analysts used and evaluated the tool, as their understanding was shaped by prior interactions rather than independent exploration. Consequently, this could influence perceived usability and effectiveness, potentially limiting the objectivity of the evaluation.

Conclusion Validity: The absence of statistical tests directly resulted from the small and homogeneous set of user stories. Thus, no statistical analysis could be performed in the evaluation, as the dataset lacked sufficient samples and variation to support meaningful statistical analysis. Another threat to conclusion validity is that LLM produces varying outputs for the same input, and the evaluation in this study did not control the temperature parameter, nor collected several responses.

7.3. Future work

So far, the quality of the designed test has been evaluated solely on its static properties (clarity, coverage, etc). Validating the efficacy of the designed test suites in detecting bugs is a complementary and essential aspect of the evaluation. Furthermore, recording and comparing the differences in the effort while adopting the tool needs new investigations.

We chose the GPT-4 model because it was the one made available to us by the public service. Possibly using different models such as (*Gemini*, 2025) and (*Claude*, 2025) could lead to diverse results. This is left as a future investigation.

Another future direction is to identify and control the factors that influence the quality of the tests. A complementary effort is to plan a more robust evaluation with a significant number and variety of user stories. Such a larger amount of input would facilitate the application of comprehensive statistical analysis.

The tool was designed to operate solely in the Redmine and Squash TM tools settings, so it must be updated to operate in different settings. Despite the setting's particularity, the core methodology may be applied to other settings. Adaptations to different environments and reusing the core methodology are important for future work.

Acknowledgment

We thank the test and business analysts who supported our project, as well as TCE-PE for providing space for our research team and offering financial resources that enabled the research and supported the involvement of faculty and students.

Contributor statement

Artur Raffael Cavalcanti (ARC): Conceptualization, Data Curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – Original Draft; Luan Accioly (LA): Conceptualization, Software; George Valença (GV): Methodology, Project administration, Writing - Review & Editing; Sidney C. Nogueira (SCN): Conceptualization, Formal analysis, Investigation, Methodology, Supervision, Writing – Original Draft, Writing - Review & Editing; Ana Carolina Morais (ACM): Project administration, Resources; Antônio Oliveira (AO): Project administration, Resources; Sérgio Gomes (SG): Project administration, Resources;

Conceptualization: SCN, ARC, LA; Data Curation: ARC; Formal analysis: SCN, ARC; Investigation: SCN, ARC; Methodology: SCN, ARC, GV; Project administration: GV, ACM, AO, SG; Software: ARC, LA; Supervision: SCN; Resources: ACM, AO, SG; Visualization: SCN, ARC, GV; Writing – Original Draft: SCN, ARC; Writing - Review & Editing: SCN, GV.

Use of AI

During the preparation of this work, the author(s) used GRAMMAR and GPT-4 in order to correct grammatical errors. After using this tool/service, the author(s) reviewed, edited, made the content their own and validated the outcome as needed, and take(s) full responsibility for the content of the publication.

Conflict of interest (COI)

There is no conflict of interest.

References

Augustine, S. (2005). *Managing agile projects*. Prentice Hall PTR.

Bahrami, M., Mansoorizadeh, M., & Khotanlou, H. (2023). Few-shot learning with prompting methods. *2023 6th International Conference on Pattern Recognition and Image Analysis (IPRIA)*, 1–5. DOI: https://doi.org/10.1109/IPRIA59240.2023.10147172.

Carvalho, G., Barros, F., Carvalho, A., Cavalcanti, A., Mota, A., & Sampaio, A. (2015). Nat2test tool: From natural language requirements to test cases based on csp. In R. Calinescu & B. Rumpe (Eds.), *Software engineering and formal methods* (pp. 283–290). Springer International Publishing.

Chinnaswamy, A., Sabarish, B. A., & Deepak Menan, R. (2024). User story based automated test case generation using nlp. In M. L. Owoc, F. E. Varghese Sicily, K. Rajaram, & P. Balasundaram (Eds.), *Computational intelligence in data science* (pp. 156–166). Springer Nature Switzerland.

Claude. (2025). https://claude.ai/

Dias Neto, A. C., Subramanyan, R., Vieira, M., & Travassos, G. H. (2007). A survey on model-based testing approaches: A systematic review. *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, 31–36. DOI: https://doi.org/10.1145/1353673.1353681.

Garousi, V., & Varma, T. (2010). A replicated survey of software testing practices in the canadian province of alberta: What has changed from 2004 to 2009? [Interplay between Usability Evaluation and Software Development]. *Journal of Systems and Software*, *83*(11), 2251–2262. DOI: https://doi.org/https://doi.org/10.1016/j.jss.2010.07.012.

Gemini. (2025). https://gemini.google.com/app

Gpt-4. (2025). https://openai.com/index/gpt-4/

Granda, M. F., Parra, O., & Alba-Sarango, B. Towards a model-driven testing framework for gui test cases generation from user stories. In: In *Proceedings of the 16th international conference on evaluation of novel approaches to software engineering - enase*. INSTICC. SciTePress, 2021, 453–460. ISBN: 978-989-758-508-1. DOI: https://doi.org/10.5220/0010499004530460.

Kushwaha, D. S., & Misra, A. K. (2008). Software test effort estimation. *SIGSOFT Softw. Eng. Notes*, *33*(3). DOI: https://doi.org/10.1145/1360602.1361211.

Leotta, M., Yousaf, H. Z., Ricca, F., & Garcia, B. (2024). Ai-generated test scripts for web e2e testing with chatgpt and copilot: A preliminary study. *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, 339–344. DOI: https://doi.org/10.1145/3661167.3661192.

Likert scale. (2006). https://conjointly.com/kb/likert-scaling/

Lins, L., Accioly, L., Nogueira, S., Valença, G., Machado, A., Lira, A., & Gomes, S. (2023). Evolução do processo de testes do tce-pe: Resultados preliminares de um projeto de bpm. *Anais Estendidos do XIX Simpósio Brasileiro de Sistemas de Informação*, 120–122. DOI: https://doi.org/10.5753/sbsi_estendido.2023.229378.

Mathur, A., Pradhan, S., Soni, P., Patel, D., & Regunathan, R. (2023). Automated test case generation using t5 and gpt-3. *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, *1*, 1986–1992. DOI: https://doi.org/10.1109/ICACCS57279.2023.10112971.

Mollah, H., & van den Bos, P. (2023). From user stories to end-to-end web testing. *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 140–148. DOI: https://doi.org/10.1109/ICSTW58534.2023.00036.

Myers, G. J., & Sandler, C. (2004). *The art of software testing*. John Wiley & Sons, Inc.

Ng, S., Murnane, T., Reed, K., Grant, D., & Chen, T. (2004). A preliminary survey on software testing practices in australia. *2004 Australian Software Engineering Conference. Proceedings.*, 116–125. DOI: https://doi.org/10.1109/ASWEC.2004.1290464.

Nogueira, S., Araujo, H., Araujo, R., Iyoda, J., & Sampaio, A. (2019). Test case generation, selection and coverage from natural language. *Science of Computer Programming*, *181*, 84–110. DOI: https://doi.org/https://doi.org/10.1016/j.scico.2019.01.003.

Nomura, N., Kikushima, Y., & Aoyama, M. (2013). Business-driven acceptance testing methodology and its practice for e-government software systems. *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, *2*, 99–104. DOI: https://doi.org/10.1109/APSEC.2013.122.

Pallavi Pandit, S. T. (2015). Agileuat: A framework for user acceptance testing based on user stories and acceptance criteria. *International Journal of Computer Applications*, *120*(10), 16–21. DOI: https://doi.org/10.5120/21262-3533.

Raharjana, I. K., Siahaan, D., & Fatichah, C. (2021). User stories and natural language processing: A systematic literature review. *IEEE Access*, *9*, 53811–53826. DOI: https://doi.org/10.1109/ACCESS.2021.3070606.

Redmine. (2006). https://www.redmine.org/

Ross, S. I., Martinez, F., Houde, S., Muller, M., & Weisz, J. D. (2023). The programmer's assistant: Conversational interaction with a large language model for software development. *Proceedings of the 28th International Conference on Intelligent User Interfaces*, 491–514. DOI: https://doi.org/10.1145/3581641.3584037.

S Al- Tarawneh, E., & Al-Tarawneh, L. k. (2022). Introducing comprehensive software quality model for evaluating and development e-government websites in jordan using fuzzy analytical hierarchy process. *Webology*, *19*(1), 890–902.

Sivaji, A., Abdullah, A., & Downe, A. G. (2011). Usability testing methodology: Effectiveness of heuristic evaluation in e-government website development. *2011 Fifth Asia Modelling Symposium*, 68–72. DOI: https://doi.org/10.1109/AMS.2011.24.

Sommerville, I. (2007). *Software engineering*. Addison-Wesley. http://books.google.ie/books?id=B7idKfL0H64C

Squash tm. (2025). https://tm-en.doc.squashtest.com/latest/

Appendix A

Answers for Q4

- **Test analyst 1:** "I liked the integration it has with Squash TM and Redmine, the formatting of the test case (this helps a lot since it creates the steps and prerequisites). I liked how it generates multiple test cases for the same story, considering both positive and negative scenarios. What I didn't like is that sometimes the tool doesn't generate test cases for all the stories."
- **Test analyst 2:** "The tool manages to create some scenarios, but it presents many errors before doing so. I believe it requires some improvement in certain areas."
- **Test analyst 3:** "Very useful."

Answers for Q8

- **Test analyst 1:** "In my opinion, the tool effectively explores test creation based on what is written in the stories. This highlights the need to continuously improve the content of the stories themselves. Just as a suggestion, it could enhance the created prerequisites, perhaps by providing more information and training to better identify these conditions."
- **Test analyst 2:** "I'm not sure if it could be improved, but in some scenarios, only the basics are created, even when the story is very detailed."
- **Test analyst 3:** "I recently started using it, and I'm still testing it in my activities."