AGORA: An Approach for Generating Acceptance Test Cases from Use Cases

Gabriele De Vito, Gabriele Vassallo, Fabio Palomba, Filomena Ferrucci
Software Engineering (SeSa) Lab – University of Salerno, Salerno, Italy
gadevito@unisa.it, vaxgab@gmail.com, fpalomba@unisa.it, fferrucci@unisa.it

***Abstract*—This paper introduces AGORA, an innovative approach that leverages Large Language Models to automate the definition of acceptance test cases from use cases. AGORA consists of two phases that exploit prompt engineering to 1) identify test cases for specific use cases and 2) generate detailed acceptance tests cases. AGORA was evaluated through a controlled experiment involving industry professionals, comparing the effectiveness and efficiency of the proposed approach with the manual method. The results showed that AGORA can generate acceptance test cases with a quality comparable to that obtained manually but improving the process efficiency by over 90% in a fraction of the time. Furthermore, user feedback indicated high satisfaction with using the proposed approach. These findings underscore the potential of AGORA as a tool to enhance the efficiency and quality of the software testing process.**

***Index Terms*—User Acceptance Testing; Large Language Models; Automated Software Engineering.**

I. Introduction

User Acceptance Testing (UAT) is a critical phase in the software development lifecycle [4], as it verifies that the final product adheres to the requirements and expectations of stakeholders. Use cases describing the interactions between the user and the system are commonly used to produce UATs.

Recently, the advent of Large Language Models (LLM) [14] has opened new perspectives in automating natural language-related tasks, including generating UATs. Nonetheless, the use of LLMs in this field is not without challenges. Issues arise from the non-determinism of LLMs, the context size limit, and the need to produce acceptance tests written in a standard format [21] that cover all event flows of the use case. This paper presents an LLM-based approach for UAT, named AGORA, which aims to overcome these challenges through prompt engineering and hyperparameter tuning. AGORA consists of two main phases: 1) Listing test cases and 2) Writing the UAT details. Both phases leverage LLMs to analyze use cases and produce UATs. The first phase focuses on identifying test cases, while the second phase is dedicated to the detailed generation of each test case. We developed a Web Tool, implementing AGORA, and used it to validate the approach conducting a controlled experiment with an Italian IT company. The study involved seven professionals. One served as an oracle, while the other six were randomly divided into two groups to produce the UATs: an experimental group using AGORA and a control group performing manually. Both groups were assigned the same use cases. The oracle assessed the quality of the produced UATs at the final stage, unaware of the group that produced them. The results highlighted that AGORA produces UATs as effectively as manual techniques but much faster. User feedback also reflects great satisfaction with AGORA. All this underscores the potential of the approach to enhance software testing efficiency and overcome the hurdles of manual UAT creation.

Structure of the paper. The remainder of the paper is organized as follows. Section II provides background and motivation of our work. Section III presents the AGORA approach, while the empirical experiment employed to validate it is described in Section IV. Section V report the results, while the main findings and the threats to the validity discussion are provided in Section VI. Section VII covers the related work. Conclusions and future work conclude the paper.

II. Background

This section introduces the essential information on LLMs and the motivation behind our study.

A. Large Language Models

An LLM is a sophisticated deep neural network trained on vast textual datasets, such as books, source code, and web pages. This training enables the model to learn complex linguistic structures and relationships, making it capable of generating coherent, grammatically correct, and human-like text [6, 10, 21]. LLMs are used in applications like automatic translation, text summarization, content generation, and question answering [3], impacting numerous sectors. Several ”emergent abilities” have been discovered in LLMs [21], including ”in-context learning,” ”following instructions,” and ”step-by-step reasoning.” ”In-context learning,” introduced by GPT-3, allows the model to generate expected outputs from provided instructions or task demonstrations without further training. ”Following instructions” enables LLMs to follow directions for new tasks without explicit examples [19]. ”Step-by-step reasoning” allows LLMs to solve complex tasks involving multiple reasoning steps, such as arithmetic or common-sense tasks. These characteristics suggest LLMs can be used to interpret and analyze use cases to produce UATs automatically. Notable LLMs include OpenAI’s GPT [3] and Meta’s LLaMA [16]. This study used GPT-4 [13], but the AGORA approach is flexible enough to use different LLMs.

B. Motivation of our work

LLMs have several limitations despite their remarkable features. They cannot make logical inferences beyond their training data, leading to nonsensical or incorrect outputs, especially with unclear or ambiguous prompts [2]. LLMs are also sensitive to input formulation [11]; slight variations can cause significantly different outputs, complicating reliable control [11]. Additionally, the amount of information LLMs can process simultaneously is limited and varies by model. For instance, GPT-4 can handle up to 8,192 tokens in a single interaction [13]. Consequently, using LLMs to analyze use case models and generate UATs can lead to incorrect results. To better understand these aspects, let us look at some examples.

Prompt no. 1: Generating UATs from a Use Case

Provide the user acceptance tests for the following use case:
US-1.02: Add a Measurement
Description: The user adds a spectral index measurement to historical agricultural land data.
Actors: Farmer
Main Scenario

1) *The user clicks on "New Measurement"*
2) *The system displays a form to select the spectral index*
3) *The user selects the desired spectral index*
4) *The system displays the parameters related to the given index*
5) *The user fills out the form*
6) *The user presses the "Save" button*
7) *The system adds the new measurement for the selected index*
8) *The system displays the message "Save successful"*
9) *The system displays the updated list of measurements*

Alternative Flows Alternative Flow No. 01

4a *At step 4, the user presses the "Cancel" button*
5a *The system closes the form*

Exceptions Exception Flow No. 01

7a *At step 7, the save fails*
8a *The system displays the message "Save failed error 564"*

Fig. 1: A simple prompt to produce UATs.

LLM output for prompt no. 1

Title: Verify Successful Addition of a Spectral Index Measurement
Test Steps:

1) *Click on the "New Measurement" button.*
2) *Verify that the system displays a form to select the spectral index.*
3) *Select a desired spectral index from the available options.*
4) *Verify that the system displays the parameters related to the given index.*
5) *Fill out the form with the necessary measurement details.*
6) *Click the "Save" button.*
7) *Verify that the system saves the new measurement for the selected index.*
8) *Verify that the system displays the message "Save successful".*
9) *Verify that the system displays the updated list of measurements, including the newly added measurement.*

Fig. 2: LLM response for the prompt provided in Figure 1.

Second LLM output for prompt no. 1

Title: Add a New Spectral Index Measurement
Objective: Verify that the user can successfully add a spectral index measurement to the historical measurements of agricultural land.
Steps:

1) *Navigate to the "New Measurement" section.*
2) *Select a spectral index from the provided options.*
3) *Fill out the form with the required parameters.*
4) *Click the "Save" button.*

Test Case 2: Cancel Adding Measurement
Objective: Verify that the user can cancel the process of adding a new measurement.
Steps:

1) *Navigate to the "New Measurement" section.*
2) *Select a spectral index and start filling out the form.*
3) *Click the "Cancel" button.*

Fig. 3: Second LLM response for the prompt provided in Figure 1.

An input prompt like the one in Figure 1 generates the output in Figure 2, which lacks features such as ID, description, precondition, actors, event flow coverage, steps, inputs, and expected results. Re-executing the same prompt yields the output in Figure 3, which includes "Title" and "Objective" and captures the alternative flow but not the error flow, highlighting the approach's non-determinism and lack of unique test format or predictable flow coverage. In verbose scenarios, such as a 12,460-character use case with 19 event flows, the output is truncated when the LLM's token limit is reached. Complete examples are available in our online repository [17].

III. The AGORA Method

AGORA is an approach that unfolds in two primary phases (see Figure 4): 1) Listing test cases and 2) Writing the test case details. Both phases leverage LLMs to analyze use cases and produce UAT.

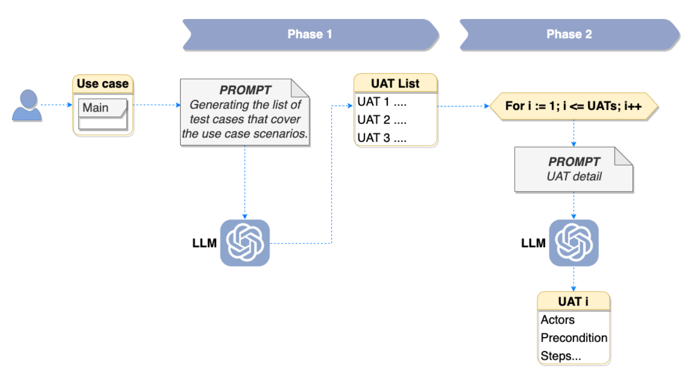

Fig. 4: AGORA Process.

In the first phase, AGORA employs an LLM to examine use cases, pinpointing testable flows through sophisticated semantic analysis to grasp context and intent. It outputs a JSON-formatted structured list of test cases detailing identifiers, descriptions, flow types (main, alternative, exception), inter-use case relationships, and explicit presence within the use case. The second phase of AGORA takes the list of identified test cases. It proceeds with the detailed generation of each test case, including the preconditions, involved actors, and the sequence of test steps with specific inputs and expected results. This phase is crucial to ensure the test cases are complete, accurate, and ready to be executed. We utilized GPT-4 to implement our approach, although AGORA is compatible with other models. Additionally, we created a user-friendly web application to streamline test case generation and facilitate AGORA's empirical evaluation, catering to users without deep expertise in language models or software testing.

A. Phase 1: Listing test cases

The first phase of the AGORA approach is crucial for outlining the functional perimeter of the UAT. This phase directly addresses the challenges posed by the context limits and the non-determinism of GPT-4. The context limitation implies that GPT-4 can process a limited number of tokens (8192, with a maximum of 4096 for the output)[1]. In the presence of long or complex inputs, exceeding this limit would lead to incomplete and insufficiently detailed responses. Moreover, the model's non-determinism can lead to variable responses even with the same input. All this could fail to ensure systematic and complete functional coverage. Finally, there is a risk that the model may generate test cases that do not match the anticipated scenarios, wasting resources and time. In Section II-B, some examples show the issues previously described.

A well-designed prompt has mitigated the described issues. We exploited the following guidelines:

1) Generate a test case for the main flow and each alternative or exception flow in the use case.
2) Structure test cases in JSON format, following a given example.
3) Exclude inexplicitly stated scenarios in the use case.
4) Avoid additional information or formatting not requested.

These guidelines optimize GPT-4's capabilities for effective test case generation. The prompt standardizes GPT-4's output into a JSON format, listing test cases with unique identifiers, descriptions, flow types, UAT indicators, and explicit presence in the original use case. This format is easily interpretable, aiding subsequent processing and integration. Figure 5 shows the prompt used in the first phase, and Figure 6 displays an example output.

The iterative prompt strategy involved evaluating LLM answers at each step to ensure they met the original requirements, refining them with subsequent prompts to fine-tune the results. This process ensures quality and accuracy before moving to the next phase.

[1]OpenAI Documentation. https://platform.openai.com/docs/api-reference

Prompt List Tests

Act as an experienced software engineer in test engineering. Produce an acceptance test using the provided use case, following these guidelines:
*** Guidelines **:*

1) *Generate a single test case for the main flow of the provided use case.*
2) *Generate a single test case for each alternative flow or exception explicitly reported in the use case.*
3) *Provide the test case in JSON format, following the structure of the provided example.*
4) *If alternative or error scenarios are not explicitly specified in the provided use case, provide only the test case for the main flow.*
5) *Do not include additional information or markdown formatting in your response JSON.*

Here is an example of how to structure the JSON:
{”Tests”: [{”Id”: ”Acceptance Test ID, in the format TA-three-digit progression”, ”Description”: ”brief but explanatory description of the test case, for example Personal Data Entry”, ”SC”: ”P if main flow, FA if alternative flow, FE if exception flow”, ”SS”: ”S if the flow simply includes another use case and does not add further specific steps, N otherwise”, ”ES”: ”S if the flow is explicitly present in the use case or if it is the main scenario, N otherwise”, ”UC”: ”use case ID”}]}
Answer only with the requested JSON without markdown and without adding other information.

Fig. 5: Test list prompt.

Example result for the test list prompt.

{”Tests”: [{”Id”: ”TA-001”, ”Description”: ”Access the ’Settings’ section in the ’Configuration’ interface”, ”SC”: ”P”, ”SS”: ”N”, ”ES”: ”S”, ”UC”: ”US0.001”}, {”Id”: ”TA-002”, ”Description”: ”Initiate ’Configure the parameters of the Settings section’ after filling in the blank fields”, ”SC”: ”FA”, ”SS”: ”S”, ”ES”: ”N”, ”UC”: ”US0.001”}]}

Fig. 6: Result of the test list prompt.

B. Phase 2: Writing the test case details

The second phase of the AGORA generates detailed UAT, ensuring each test case is complete, correct, clear, and ready for execution. It aims to create unambiguous test cases that are aligned with use case scenarios and structured in a JSON format for easy understanding and implementation. The format includes:

- Precondition: Specifies the initial conditions under which the test should be executed, ensuring that the context is appropriate for the test flow.
- Actors: Lists the involved actors, ensuring that all roles necessary for the test are identified and understood.

- Steps: Details the test steps, including inputs and expected results, to provide step-by-step guidance for test execution.

Using specific prompts, GPT-4 completes test case details for each acceptance test ID, ensuring test steps are accurate and reflect the use case. This phase can be parallelized, as each test case is independent, thus optimizing efficiency and speed.

> **Prompt List Tests**
>
> *I am providing you with a use case and the identified acceptance test IDs. Act as a software engineer experienced in test engineering to complete the acceptance test related to the use case. Respond with a well-formed JSON as in the following example.*
> *Example:*
> *{"Precondition": "if SC='P' precondition of the use case, otherwise insert the steps of the main flow to be executed", "Actors": "list of actors of the use case", "Test": [{"Step": "increasing numbering of the step performed", "Input": "input", "Result": "expected result of the step"}]}*
> *Answer only with the JSON I requested, without adding any other information.*
> ***** Start Use Case*
> *{use_case}*
> ***** End Use Case*
> *{uat_list_from_phase1}*
> *Provide the user acceptance test with ID = {ID_UAT}*

Fig. 7: Prompt for test case details.

An iterative prompt engineering approach, similar to Phase 1, was used with progressive refinements. The final Phase 2 prompt is shown in Figure 7, and an example output is in Figure 8.

> **UAT detail example**
>
> *{"Precondition": "At least one measurement exists in the database", "Actors": ["GP", "Relative"], "Test": [*
> *{"Step": 1, "Input": "The actor accesses the 'Measurements' section", "Result": "The system displays an interface where it is possible to view for each parameter the measurements related to the last update"}, {"Step": 2, "Input": "The actor uses the graph viewing feature related to one of the detected parameters", "Result": "The system opens a popup with the available graphs for the selected parameter"}, {"Step": 3, "Input": "The actor navigates the interface", "Result": "The system allows the navigation of the graphs and the display of the measurements in graphic form"}] }*

Fig. 8: Result for producing the UAT details.

C. Optimization of Determinism

Determinism in the context of this work refers to the ability of the LLM to provide consistent and predictable results in response to similar or identical inputs. This aspect is crucial for UAT. For optimizing determinism, it is necessary to operate in multiple directions. First and foremost, it is essential to carefully select and configure the hyperparameters of GPT-4 that influence the generation of responses, in particular:

- **Temperature:** It regulates the model's predictability and originality, with values from 0 to 1. A temperature near 1 increases randomness and diversity, while a near 0 makes the output more deterministic and predictable.
- **Top_p:** It refines text generation in LLMs by selecting words with high cumulative probability, balancing creativity and consistency.
- **Best_of:** It generates multiple server-side completions, selecting the 'best' response based on the lowest logarithmic probability per token.
- **Frequency_penalty:** It controls the model's tendency to repeat predictions by decreasing the likelihood of previously generated words based on their frequency in the prediction.
- **Presence_penalty:** It promotes originality by reducing word repetition, unlike the frequency penalty, which ignores prior term usage.

Therefore, according to OpenAI documentation, we set the temperature, presence penalty, and frequency penalty to 0, while best and top p to 1, making the model's responses more deterministic. Secondly, prompts must be carefully designed with clear, detailed instructions to guide GPT-4 in generating specific and relevant outputs, reducing ambiguity and increasing consistency. Examples of desired outputs and structured templates were provided to stabilize responses, following the "show, do not tell" approach [7]. Finally, after each output generation, results were analyzed for consistency and adherence to requirements, identifying non-determinism patterns and adjusting prompts as needed.

D. AGORA Tool

We developed a Web tool using the Streamlit Python framework [2] and GPT-4 API [3]. The tool analyzes GPT-4 responses and displays UATs on a web page. The tool features an intuitive interface for entering use case scenarios and generating corresponding UATs, manipulating the JSON format produced by GPT-4 into a user-friendly display.

E. Replication Package

The replication package of our study is publicly available in the online GitHub repository [17] and includes the following: i) the source code of the AGORA tool, ii) the use case model used for the empirical experiment, iii) the UATs produced by the control and experimental groups, and iv) the results of the experiment. The use of the AGORA tool is licensed under an International Creative Commons Attribution 4.0 license.

IV. Empirical Experiment

This section details our empirical experiment to evaluate the proposed approach, covering the research question, hypotheses, experiment context, variables, design, and analysis plan.

[2]Streamlit Framework. https://streamlit.io/
[3]OpenAI Documentation. https://platform.openai.com/docs/api-reference

A. Research question and hypothesis

AGORA aims to help test engineers produce high-quality UATs faster than the manual approach. Thus, we propose to answer the following research question:

> *Can AGORA support software engineers by automating the production of UATs from use cases?*

The research question has been outlined through the formulation of the following hypotheses:

- **H1**: AGORA produces qualitatively better UATs than the manual approach.
- **H2**: AGORA speeds up UAT production from use cases.

B. Context of the Experiment

To address the research question and assess the proposed approach, we conducted a controlled experiment with six professionals from an Italian IT company, Kiranet, and a professional consultant. Kiranet, established in 2002 with over 50 employees, regularly uses UML in its software development process. The company's project manager selected test engineers with a good understanding of UML and UAT and similar experience levels. The consultant has over 25 years of software engineering experience. The experiment used ten use cases (available online in the repository [17]) from a telemedicine microservice, FIDTCT, provided by Kiranet, varying in complexity, quality, and linguistic styles.

C. Variable Selection

The following dependent variables have been identified for the experiment:

Quality: High-quality UATs must meet the following criteria:

- *Completeness*: It ensures UATs cover all the use case flows, with a test case for every non-trivial flow.
- *Clarity and Understandability*: It ensures test cases are clear and unambiguous and the steps to be executed are correct.
- *Correctness*: It ensures test cases are formally and semantically correct, verifying that actors, preconditions, postconditions, inputs, and outputs are accurate and consistent with the use case requirements.

TABLE I report the criteria, elements and evaluation formulas for *Quality* variable.

Time: This variable measures the time required to produce the UATs for each use case.

The independent variable is the **approach used**, which refers to using AGORA or the manual approach.

Criterion	Elements to asses	Evaluation
Completeness (COM)	• Number of tested flows (FT) • Number of use case flows (NF)	$COM = FT/NF$
Clarity and Understand-ability (CeC)	CheckList: • **Language (CC1):** Is the test case language simple and clear? • **Instructions (CC2):** Are the instructions clear and easy to follow? • **Consistency (CC3):** Are consistent terms used throughout the test case? • **Ambiguity (CC4):** Does the test case avoid ambiguities?	*Checklist items are assigned a value of 0 if the criterion is not met and a value of 1 if the criterion is met.* $CeC = \sum_{EC=CC1}^{CC4} EC/NE$
Correctness (COR)	CheckList: • **Actors (CO1):** Do the test cases accurately identify the actors and their interactions with the system? • **Precondition (CO2:** Is the test case's precondition consistent with the anticipated use case? • **Postcondition (CO3):** Is the postcondition verified to ensure the system's expected state post-test execution? • **Input (CO4)** Are all input actions from the use case correctly reported in the test case? • **Output (CO5):** Are all expected outputs correctly reported in the test case for the given inputs? • **Validity concerning requirements (CO6):** Does each test case accurately reflect the use case requirements? • **Scenarios (CO7):** Have the essential scenarios been tested for correct requirements validation?	*Checklist items are assigned a value of 0 if the criterion is not met and a value of 1 if the criterion is met.* $COR = \sum_{EC=CO1}^{CO7} EC/NE$
	QUALITY VALUE:	$Mean(COM, CeC, COR)$

TABLE I: Quality evaluation.

D. Experiment Design

We chose the consultant as the oracle and randomly assigned test engineers to the experimental or control groups, keeping their assignments confidential. Both groups were tasked with producing UATs for given use cases; the experimental group used AGORA, while the control group did it manually. Participants were informed about data confidentiality and their right to leave the experiment at any time. The experimental group received one-hour training on using AGORA to generate UATs from use cases. Both groups were given a Word template for writing UATs and shown how to track execution times before working independently.

Question	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
AGORA is easy to use					
AGORA is useful for writing UATs					
I would use AGORA for writing UATs					
I would like to try similar tools for other software engineering tasks.					

TABLE II: Questionnaire on the experience with AGORA

The test cases produced by the two groups were anonymously evaluated by the oracle, based on the criteria indicated in Section IV-C, using the form shown in Figure 9 to record the scores for the various criteria.

At the end of the experiment, we asked the experimental group to complete a satisfaction questionnaire (shown in Table II) about the AGORA tool, using a Likert scale from 1 (Strongly Disagree) to 5 (Strongly Agree).

Use Case no. X																		
		No of SCENARIOS	Criteria															
			Completeness	Clarity and Understandability					Correctness								Time (seconds)	
		No. of UAT		CC1	CC2	CC3	CC4	Value	CO1	CO2	CO3	CO4	CO5	CO6	CO7	Value		MEAN
Group no. 1	Participant 1																	
	Participant 2																	
	Participant 3																	
	MEAN																	
Group no. 2	Participant 4																	
	Participant 5																	
	Participant 6																	
	MEAN																	

Fig. 9: Form filled out by the oracle for each use case.

E. Analysis Plan

To verify hypotheses H1 and H2, we decided upon the following statistical analysis methods:

- **Hypothesis H1 (Quality of Acceptance Tests)**: To analyze the quality of UATs from the experimental and control groups, we will calculate descriptive statistics (mean, median, standard deviation) for scores assigned by the oracle on completeness, correctness, and clarity. The Mann-Whitney U test will compare the mean quality percentages between the groups at a 0.05 significance level to determine if there are statistically significant differences. The null hypothesis states no difference in average quality scores between the groups, while the alternative hypothesis suggests a significant difference.
- **Hypothesis H2 (Production Time of Acceptance Tests)**: We planned to calculate descriptive statistics (mean, median, standard deviation) for the time taken by the experimental and control groups to produce UATs. We will use the Mann-Whitney U test with a significance level of 0.05 to compare the average times. This test assesses whether there is a significant difference in the time taken between the AGORA and control groups. The null hypothesis is that there is no difference, while the alternative hypothesis is that the AGORA group took significantly less time.

Finally, we planned to calculate the average scores for each questionnaire question II to assess user satisfaction with AGORA. The questionnaire collects qualitative feedback on users' experiences, offering insights for future improvements.

V. Analysis of the Results

This section presents the empirical experiment results. We analyzed the collected data (available online in the repository [17]) to assess hypotheses H1 and H2, focusing on UAT quality and production time.

A. Quality of Acceptance Test Cases (Hypothesis H1)

The oracle evaluated the quality of UATs based on the criteria of completeness, clarity, understandability, and correctness as described in Section IV-C. Table III presents the results obtained for the AGORA group, while Table IV shows those for the control group. We evaluated completeness based on use case flow coverage, with both groups achieving 100%. Clarity and understandability were assessed using the criteria in Section IV-C, with both groups averaging 83%. Correctness, based on actors, preconditions, postconditions, inputs, outputs, and requirement validity, was 80% for both the control and AGORA groups.

Use Case	Completeness	Clarity and Understandability	Correctness	Average	Time(s)
1	100%	100%	71%	90%	31.96
2	100%	100%	71%	90%	22.83
3	100%	75%	71%	82%	30.42
4	100%	75%	86%	87%	33.00
5	100%	100%	100%	100%	31.46
6	100%	100%	86%	95%	22.78
7	100%	75%	86%	87%	23.16
8	100%	50%	71%	74%	28.79
9	100%	50%	71%	74%	35.56
10	100%	100%	86%	95%	32.19
Average	100%	83%	80%	88%	29.22

TABLE III: Experimental group UAT results

Use Case	Completeness	Clarity and Understandability	Correctness	Average	Time(s)
1	100%	100%	67%	89%	324.00
2	100%	100%	67%	89%	718.33
3	100%	75%	62%	79%	217.33
4	100%	75%	81%	85%	607.33
5	100%	100%	95%	98%	446.33
6	100%	100%	86%	95%	254.00
7	100%	75%	81%	85%	233.00
8	100%	50%	81%	77%	557.33
9	100%	50%	81%	77%	281.33
10	100%	100%	95%	98%	260.33
Average	100%	83%	80%	87%	389.93

TABLE IV: Control group UAT results

Nevertheless, the AGORA group scored better in 6 use cases (i.e., 1-5 and 7), performing worse only in 3 out of 10 cases (i.e., 8-10). The overall average quality for the control group was 87%, while for the AGORA group, it was 88%. Finally, the standard deviation for quality for the control group was 0.08, while for the AGORA group, it was 0.09. Then, we analyzed the results using the Mann-Whitney U test to compare the average quality scores between the control and AGORA groups. The test accounted for ties in the data, applying a correction. The p-value of 0.9093 indicates no statistically significant difference between the two groups in test case quality.In other words, there is not enough evidence to reject the null hypothesis, which asserts that there is no difference between the groups. The very high p-value, 90.93%, indicates that if we were to reject the null hypothesis, there would be a high probability of making a Type I error, that is, rejecting a true null hypothesis. Furthermore, the test statistic Z obtained is -0.1139, well within the acceptance range for a 95% confidence level (between -1.96 and 1.96). This result reinforces the idea that the groups have no significant difference. Regarding the effect size, the observed standardized effect size, $Z/\sqrt{(n_1+n_2)}$, is 0.025, indicating a minor difference between groups. The common language effect size, $U_1/(n_1 n_2)$, is 0.48, showing a 48% chance that a test case from the AGORA group is of higher quality than one from the control group, suggesting no practical difference.

B. Acceptance Test Case Production Time (Hypothesis H2)

The time to produce UATs was measured in seconds. The control group took 3,899.33 seconds total (389.93 seconds per use case), while the AGORA group took 292.16 seconds total (29.22 seconds per use case). The medians were 30.94 seconds for AGORA and 302.67 seconds for the control group, with standard deviations of 4.67 and 180.17, respectively.We used the Mann-Whitney U test to compare the average times of the control and AGORA groups, and we found a statistically significant difference. Indeed, the p-value obtained from the test is extremely low at 0.00001083, indicating a 0.0011% chance of a Type I error. This result strongly supports the alternative hypothesis of a significant difference in the time taken by the two groups to produce test cases. The test statistic, Z, is 4.4, outside the 95% acceptance range (-1.96 to 1.96), confirming the significance of the observed difference. Additionally, the U value of 100 is outside the 95% acceptance range (24 to 76), further reinforcing the significance of the differences. Regarding the effect size, the observed standardized value,

$Z/\sqrt{(n_1 + n_2)}$, is significant (0.84), indicating a considerable difference between the control and AGORA groups. Additionally, the common language effect size, $U_1/(n_1 n_2)$, is 1, suggesting a very low probability that a random control group value exceeds a random AGORA group value.

C. User Feedback on the Use of AGORA

The questionnaire results II are shown in Table V. Participants rated AGORA highly, with averages of 4.3 for ease of use and 4.0 for usefulness, future use, and interest in similar tools. Responses showed minimal deviation (0 to 0.58) and a median of 4 for all questions, indicating consensus.

Question	Average	Median	Std. Dev.
AGORA is easy to use	4.3	4.0	0.58
AGORA is useful for writing acceptance tests	4.0	4.0	0.00
I would use AGORA for writing acceptance tests	4.0	4.0	0.00
I would like to try similar tools for other software engineering tasks	4.0	4.0	0.0

TABLE V: Feedback on the experience with AGORA

VI. Discussion, Implications, and Future work

This section discusses the results reported in Section V, their implications, and the limitations that may have influenced our findings and mitigation strategies.

A. Discussion

Hypothesis H1 suggested that AGORA could produce better UATs than the manual approach, but the experimental results did not support this statistically. AGORA and the control group showed comparable quality levels. AGORA generally scored better in correctness (6 out of 10 use cases) but was less effective in inferring input and output data in 3 use cases (8-10). This result could be due to insufficiently detailed use cases, limiting AGORA's ability to infer necessary data using GPT-4. For example, in use case no. 3, it generically mentions "details" and "fields" without specifying what is being searched for and displayed:

> **Use Case no.3**
>
> *Main scenario*
>
> 1) *The actor clicks "view detail" for a given threshold.*
> 2) *The system searches for the threshold details.*
> 3) *The system displays the threshold details in a page with editable fields.*

The control group could not precisely define input and output data for the same use case, only reporting test steps. However, test engineers correctly identified input data in other cases (use cases 8-10), unlike AGORA. This result suggests that the quality of automatically generated test cases depends on the detail of the starting use cases. The lack of significant difference might also be due to software engineers' familiarity with the use cases and application domain, allowing them to fill in missing information. Despite these results, AGORA is a promising tool for refining generated UATs. Automating test case generation can identify gaps in use cases and encourage adding missing details, improving overall quality. For hypothesis H2, results showed AGORA significantly reduced the time to produce UATs, confirming the effectiveness of automation. This efficiency can positively impact the software development lifecycle by reducing release times and allowing engineers to focus on complex tasks. Feedback from a questionnaire indicated high user satisfaction with AGORA, valuing its ease of use and time savings despite the quality of automatically generated test cases not surpass manual ones.We can, therefore, answer the research question as follows:

> *AGORA automates the production of acceptance test cases from use cases, enhancing efficiency, but the quality of the tests depends on the completeness and detail of the use cases.*

B. Threats to Validity

This section examines potential limitations that could have biased our results and mitigation strategies.

A threat to internal validity includes participants' bias and the Hawthorne effect, where awareness of being in an experiment alters responses. To mitigate this, we hid the hypothesis, ensured anonymity, divided participants into two groups, and shortened the study duration. Another internal validity threat is including more experienced individuals in the experimental group. We asked the IT company to provide participants with similar UML and test management experience. Additionally, we mitigated oracle bias by selecting a consultant with significant software engineering experience. Future research could involve more experts as oracles to compare evaluations of produced UATs. External validity threats include the selection of use cases. We chose use cases of varying complexity and linguistic style, but future research should include more use cases from different domains to improve generalizability. A threat to the validity of conclusions is the choice of statistical methods. We used the Mann-Whitney U test, a non-parametric test suitable for small, non-normally distributed samples, and verified its assumptions before application. We also used a correction for ties when necessary. Additionally, we addressed the non-determinism of the LLM by fine-tuning GPT-4's hyperparameters and giving clear formatting instructions, limiting the LLM's creativity for consistent responses.

VII. Related Work

Automatically producing UAT is challenging, with various proposed approaches (see Table VI). Many authors suggest using NLP, often limited to specific domains or predetermined input formats. Nebut et al. [12] propose automating system test scenario generation from UML-based use cases with contracts, but their manual-intensive approach may struggle with scalability in complex systems. Carvalho et al. [5] develop NAT2TEST, generating test cases from Controlled Natural Language requirements for Data-Flow Reactive Systems using formal models like Software Cost Reduction, and suggest future enhancements for performance optimization and hybrid system support. Yue et al. [20] introduce Restricted Test Case Modeling to convert natural language test cases into executable tests using a tool, aToucan4Test, validated with two industrial case studies. However, the study imposes restrictions on requirements and lacks performance analysis and broader generalizability. Goffi et al. [9] develop Toradocu, leveraging Javadoc comments and NLP to generate test oracles and conditional expressions for Java programs, enhancing defect detection. The prototype has limitations with complex conditions and focuses on @throws tags. Silva et al. [15] use Colored Petri Nets in NAT2TEST to generate test cases from natural language requirements, addressing state explosion but not completeness and consistency. Allala et al. [1] integrate Model-Driven Engineering with NLP to convert user requirements into test cases, potentially reducing manual generation. The research is in its initial phase with limited validation. Fischbach et al. [8] use NLP in SPECMATE to automate 56% of test case generation from agile acceptance criteria, relying on language patterns and lacking domain independence. Fischbach et al. [8] automate test case generation from agile acceptance criteria using NLP in SPECMATE, automating 56% of test cases. The method depends on language patterns and is not domain-independent.

Study	Main Features	Limitations
[12]	Formalizes use case descriptions and generates test cases.	Requires preconditions and post-conditions in a specific format.
[5]	Uses Data-Flow Reactive Systems (DFRS) requirements.	Limited to requirements in controlled natural language.
[20]	Generates test cases from an RTCM language.	Limited to the writing style of test cases imposed by RTCM.
[9]	Uses Javadoc comments to complete test cases.	Comments must follow a specific pattern.
[15]	Explores test case generation using Petri Net simulation.	Interpretation of Colored Petri Nets can vary.
[1]	Generates test cases from use cases or user stories.	Requires a specified format for use cases or user stories.
[8]	Uses recursive dependency matching to formulate test cases.	Requires advanced knowledge of dependency matching.
[18]	Requirements compatible with RUCM specifications.	Limited to the use of RUCM specifications.

TABLE VI: Summary table of related works

AGORA stands out from related works by accepting requirements in free natural language, eliminating rigid writing patterns, and allowing natural documentation. Its versatility suits diverse applications, simplifying UAT generation and reducing production time. AGORA eases the cognitive load on engineers and testers, avoiding the need to learn new languages or modify workflows, thus enhancing efficiency. Integration into existing workflows is seamless, requiring no significant changes in software development processes.

VIII. Conclusion

In this study, we introduced AGORA, a novel approach for automatically generating UATs from requirements in natural language. Leveraging LLMs, AGORA interprets unstructured requirements and generates accurate UATs, optimizing time and resources, reducing human errors, and increasing testing efficiency. Initial findings show that AGORA produces UATs of similar quality to those created by experts and significantly improves test case generation efficiency. Moreover, AGORA also exhibits a marked increase in test case generation efficiency. The approach can be easily integrated into existing software development processes, offering a considerable competitive advantage and facilitating adoption in the Industry. To improve AGORA, we will enhance prompts or fine-tune the LLM for ambiguities and missing information, integrate a feedback system for refining test cases, and develop guidelines for optimized use cases. Enhancing user collaboration through better interfaces is also a key objective.

References

[1] S. C. Allala *et al.*, "Towards transforming user requirements to test cases using mde and nlp," in *IEEE COMPSAC*, vol. 2, 2019, pp. 350–355.

[2] E. M. Bender, T. Gebru *et al.*, "On the dangers of stochastic parrots: Can language models be too big?" in *ACM Conference on Fairness, Accountability, and Transparency*, 2021, p. 610–623.

[3] T. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. L. et al., Ed., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[4] B. Bruegge and A. H. Dutoit, "Object–oriented software engineering. using uml, patterns, and java," *Learning*, vol. 5, no. 6, p. 442, 2009.

[5] G. Carvalho *et al.*, "Nat2test tool: From natural language requirements to test cases based on csp," in *Software Engineering and Formal Methods*. Cham: Springer International Publishing, 2015, pp. 283–290.

[6] M. Chen *et al.*, "Evaluating large language models trained on code," *arXiv*, 2021.

[7] S. Ekin, "Prompt Engineering For ChatGPT: A Quick Guide To Techniques, Tips, And Best Practices," 2023. [Online]. Available: https://www.techrxiv.org/articles/preprint/Prompt_Engineering_For_ChatGPT_A_Quick_Guide_To_Techniques_Tips_And_Best_Practices/22683919

[8] J. Fischbach *et al.*, "Specmate: Automated creation of test cases from acceptance criteria," in *IEEE ICST*, 2020, pp. 321–331.

[9] A. Goffi *et al.*, "Automatic generation of oracles for exceptional behaviors," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 2016, p. 213–224.

[10] E. Kasneci, K. Sessler *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, p. 102274, 2023.

[11] Y. Lu *et al.*, "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity," *arXiv*, 2021.

[12] C. Nebut *et al.*, "Automatic test generation: a use case driven approach," *IEEE TSE*, vol. 32, no. 3, pp. 140–155, 2006.

[13] OpenAI, "Gpt-4 technical report," *arXiv:2303.08774*, 2023.

[14] M. Shanahan, "Talking about large language models," 2023.

[15] B. C. F. Silva, G. Carvalho, and A. Sampaio, "Test case generation from natural language requirements using cpn simulation," in *Formal Methods: Foundations and Applications*, M. Cornélio and B. Roscoe, Eds. Cham: Springer International Publishing, 2016, pp. 178–193.

[16] H. Touvron, T. Lavril, G. Izacard *et al.*, "Llama: Open and efficient foundation language models," *arXiv:2302.13971*, 2023.

[17] G. Vassallo, "Agora github repository." [Online]. Available: https://github.com/GabrieleVassallo/AutomaticUAT

[18] C. Wang, F. Pastore *et al.*, "Automatic generation of acceptance test cases from use case specifications: An nlp-based approach," *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 585–616, 2022.

[19] J. White, S. Hays *et al.*, "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," 2023.

[20] T. Yue, S. Ali, and M. Zhang, "Rtcm: A natural language based, automated, and practical test case generation framework," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. Association for Computing Machinery, 2015, p. 397–408.

[21] W. X. Zhao *et al.*, "A survey of large language models," *arXiv*, 2023.