Automatic Generation of Bug Reports Using Large Language Models: An Evaluation in a Software Institute

Lennon Chaves*
Sidia Institute of Science and Technology
Manaus, Brazil
lennon.chaves@sidia.com

Davi Gonzaga*
Sidia Institute of Science and Technology
Manaus, Brazil
davi.menezes@sidia.com

Leonardo Tiago
Sidia Institute of Science and Technology
Manaus, Brazil
leonardo.albuquerque@sidia.com

Ana Paula Silva
Sidia Institute of Science and Technology
Manaus, Brazil
silva.ana@sidia.com

Flávia Oliveira*
Sidia Institute of Science and Technology
Manaus, Brazil
flavia.oliveira@sidia.com

ABSTRACT

Context: During software development, the test team is responsible for verifying if the requirements were correctly implemented. In case the software does not behave as expected, the tester must report the bug to the development team. For the problem to be assertively and quickly, it is important that the bug report is complete, containing all the information needed to fix the bug. This study was conducted in the context of a test team that acts in a software institute, and is responsible for testing mobile devices. **Problem**: Among their activities, the test team must write bug reports. However, given a high amount of test requests, consequently there will be many bug reports, which will take more time and effort. **Goal**: With the goal of automating the bug report process, we developed a system that automatically generates bug reports based on the Text-to-Text Transfer Transformer (T5) Large Language Model (LLM). To generate a bug report, the tester must input a prompt containing a brief description of the bug. **Method**: We also performed an experiment with 8 members from the test team that write bug reports daily, so as to measure their perceptions regarding the solution we developed. The participants used the system and evaluated its outputs for the generation of 5 distinct bug reports. **Results**: Results showed a high acceptance rate for using the system within the test team, with 3 of the 5 bug types included in the experiment having 87.5% of outputs considered valid or partially valid. Furthermore, participants highlighted the ease of use, the efficacy and quality of writing in the bug reports generated by the system. However, they also noted that the system still needs to be adjusted to work for more types of issues, thus reducing the need for manual fixes. **Conclusions**: We conclude, then, that it is possible to use LLMs to automatically generate bug reports, however it is still indispensable to have a human review the report after it is generated.

KEYWORDS

Bug Report, Large Language Models, Software Testing

1 Introduction

In a software industry testing team, bug reporting is a crucial phase of the Software Development Life Cycle (SDLC), as it is in this phase that it is ensured that system failures will not be present in the version delivered to the end user [10]. To achieve this, the software goes through various tests to verify whether it complies with the specifications defined during requirements elicitation [10]. If an issue is identified in the software, that is, something that does not match the expected requirements, it is necessary to produce a bug report to inform developers about the problem in the software implementation [10, 19]. The bug report is a document that highlights to developers that there is a failure in the software that needs to be fixed. To produce a high-quality bug report, it must include some fixed fields such as: summary, description, reporter, component, operating system, and version [24]. All these fields help guide developers to understand what kind of bug it is [4].

On the other hand, although bug reporting has fixed characteristics to follow, this process is prone to failures, since the entire writing of the report is carried out manually by testers, which requires time and effort, and in a context where a large volume of bugs is reported daily, this process can delay other testing stages [22]. In this regard, some approaches in the literature aim to address this problem in a semi-automated way, such as the Template Generator [6] which generates the structure of a bug report, and Bug Builder [7] which describes how a bug report should be registered. Other approaches, such as that of Yao et al. [25] perform bug report generation in an automated way, where they proposed BugBlitz-AI, a tool designed to support the automatic bug summarization process. Other studies explore the use of chatbots and pipelines, such as Song et al. [20], who presented BURT, a chatbot that supports the creation of bug reports, and Yang Song et al. [19], who used an automated pipeline to analyze and extract relevant information from the bug.

With the rising popularity of Artificial Intelligence (AI) technology, which is constantly evolving, it is increasingly more common to use AI applications to solve problems in the Software Engineering (SE) field [18]. Given that Large Language Models (LLMs) have been the key to solve problems involving text generation in other studies [16], we decided to use this approach to overcome the problems faced by the test team, with the goal of investigating how members of a test team from the SE industry adapt to use an LLM that automatically generates bug reports based on prompts with brief descriptions of bugs.

*Both authors contributed equally in this research.

To that end, we employed the Text-to-Text Transfer Transformer (T5) model [17], which was developed by Google as a unique solution to handle multiple types of Natural Language Processing (NLP) tasks. Specifically, we fine-tuned 3 versions of the T5 model to generate bug reports for groups of distinct bug types, and developed a system that is capable of determining which version is adequate to generate the bug report based on the user's input [11].

To validate the use of the fine-tuned T5 model for reporting bugs, we performed an experiment within a software test team that reports bugs for real projects. For this investigation, we selected 8 members from this team that report bugs everyday. The participants received a sheet with bug descriptions and predefined prompts for each bug, and were instructed to first collect outputs from the system based on the provided prompts, then register these outputs on the sheet. After that, they must also evaluate the outputs based on their perception of how valid they would be in a real scenario. Lastly, participants also answered a survey with their general perception regarding the use of the LLM-based system to report bugs.

Through these experiments, our research aims to offer contributions to the LLM field as well as the software test field, introducing an approach that can optimize the process of writing bug reporting, and thus reducing the amount of time and human effort required for this activity, and allowing these resources to be allocated to other activities in the software development life cycle.

The sections of this research are divided as follows: Section 2 presents the theoretical reference this study was based on; Section 3 details the use of the T5 model for bug reporting; Section 4 discusses the strategy we used to conduct this study; Section 5 presents the results we achieved; Section 6 discusses the factors that limited this study; and lastly, Section 7 details the conclusions we reached.

2 Theoretical Reference

To our knowledge, this is the first work to approach the use of LLMs specifically to generate complete bug reports in the Verification, Validation and Test stage of the SDLC. Now we will discuss the related works that composed the foundation of this research.

2.1 Bug Report

Bugs are inherent to software engineering, as they follow the software's evolution and make the development process more expensive because they require more time, money and other resources to be fixed [12]. During the development of a software project, managing the software and the bugs that are discovered is crucial for the success of the project, and bug reports are the most efficient approach to perform this management, as affirmed by Tan et al. [21].

Chaparro [5] explains that the information present in bug reports are inputted with the intention of helping developers to fix said bugs, but they also note that not every bug report contains this information, and even when they do, the information tends to be poorly written. In their dissertation, Chaparro reinforces that important information must always be included, such as: Observed Results, Expected Results and Reproduction Steps. The author also recommends elements of speech to describe this information adequately in bug reports.

To be able to determine what makes a bug report have quality, Bettenburg et al. [3] conducted a research in which developers answered a survey regarding information they considered important and problems that they usually find in bug reports, and also reviewed complete reports in a 5-point Likert scale. Their results denote that "incorrect reproduction steps" and "incomplete information" are the biggest hurdles faced by developers.

This problem highlights the necessity of a system that helps the tester to write their bug reports, ensuring that all information considered essential is included and are written in a way that is easy to understand.

2.2 Large Language Models

After the release of ChatGPT [1] in November 2022, the field of study of AI has been the focus of discussions about technology, attracting attention not only from the academic community but also from the general public, as ChatGPT is able to talk with its users like a real person [15].

This is made possible because ChatGPT uses an LLM as its foundation, which is a deep learning model trained with massive amounts of data publicly available, with the goal of producing answers similar to those of human beings in conversations [14].

However while chatbots like ChatGPT are currently the focus of the public's eyes, since the emergence of LLMs in 2018, it has been possible to demonstrate how they are highly beneficial to many fields of study within SE, such as software implementation and maintenance, quality analysis and requirement engineering [18].

2.3 Large Language Models for Bug Reporting

An approach like ours was used in the work by Acharya and Ginde [1], which consists in transforming non-structured bug reports in structured reports based on standardized template formats, while also denoting missing information for the writer of the report, all through the use of LLMs. Their studies showed that the Qwen 2.5 [2] model had a superior capacity to detect reproduction steps when compared to other models, while Llama 3.2 [3] reached a higher accuracy to detect fields missing information. These results are indicative of the potential that their approach has to reduce the developers' manual effort and speed up the process of software maintenance as a whole.

The work by Kumar et al. [13] explores the potential of LLMs as automatic evaluators to summarize software artifacts. They conduct comparison experiments between humans and 3 LLMs to select the correct title and summary for bug reports. Their results indicate that LLMs had a good performance and did not suffer from fatigue as the human participants, showing the potential that LLMs have to act as automatic evaluators to summarize software artifacts, and thus reducing the need of effort from human evaluations.

Other works, such as the one by Sherifi et al. [18] explore the use of LLMs to automate the software test activity, so as to enhance its efficiency while reducing the need for human intervention. The framework they propose integrates LLMs in the test process to:

[1] https://chatgpt.com/
[2] https://qwenlm.github.io/blog/qwen2.5-llm/
[3] https://www.llama.com/

generate unit tests, visualize call graphs and automate the test execution and report process. Their results, achieved through multiple applications in Python [4] and Java [5], indicate that this system can work efficiently and cover a great part of the tests.

While in the topic of automation, the work by Fend and Chen [8] presents AdbGPT, an approach for automatically performing the reproduction steps for a bug through the use of LLMs. This is initially done by having AdbGPT extract the reproduction steps from the bug report, and then it uses the extracted data to guide the automatic execution. Besides automatically performing reproduction steps, AdbGPT was also used in their study to try to enhance the developers' capacity to reproduce bugs.

The works discussed above showed how LLMs were able to infiltrate in many fields within SE, with the goal of reducing the human effort and time investment needed to perform tasks related to bug reports. This way, in this work, we seek to demonstrate how the employment of an LLM-based system that automatically generates bug reports can optimize the software test stage.

3 Bug Reporting using T5 Model

3.1 Description of the Tool

Aiming to reduce the high investment of time and effort needed for the activity of writing bug reports, a system that generates bug reports based on the T5 language model was developed [11]. This system receives an input from the user, which must contain a brief description of the bug, then it processes the input to generate a complete bug report based on it.

The T5 model [17], or Text-to-Text Transfer Transformer, was used as the base because it is a model dedicated to handling different types of NLP activities. Along the development of the system, it was decided that different iterations of the model were to be implemented, each one fine-tuned to handle certain groups of bugs, as described below:

- **Alpha Model**: The Alpha model was fine-tuned to handle 6 types of issues related to the mobile device's system.
- **Beta Model**: The Beta model was fine-tuned to handle 1 type of issue related to accessing the server of the services with which the mobile device interacts.
- **Gamma Model**: The Gamma model was fine-tuned to handle 2 types of issues related to the mobile device's settings.

The distribution of types of issues for each iteration of the T5 model was made based on the complexity of the different types of bug reports, which was measured during the development of the system. This way, the Alpha model possesses the highest amount of types of issues as they possess low complexity levels, while the other models, Beta and Gamma, possess a considerably lower amount of types of issues as they possess high complexity levels. The details about the dataset size, training procedure and evaluation metrics were discussed in a previous work [11].

3.2 Structure of the Tool's Output

While bug report structures tend to vary depending on who writes them and where they are written, the development of this system used bug reports from the test team in which this study was conducted. Initially, these reports possessed much information specific to the context of the test team, as well as additional data of the test environment in which the bugs were found, so the reports were treated to contain only the fields considered important for the model's learning, as described below:

- **Title**: A brief description of the bug, usually within a single phrase.
- **Problem**: A detailed description of the bug as a whole.
- **Preconditions**: Any conditions that need to be met before performing the bug's reproduction steps.
- **Reproduction Steps**: A step-by-step guide to recreate the scenario in which the bug was found.

3.3 Guide to Use the Tool

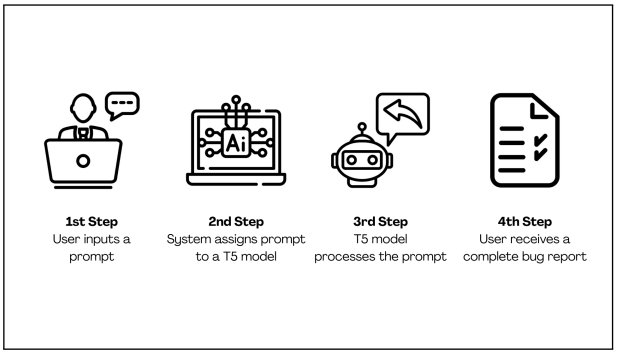

Figure 1: Process of the user interacting with the system

The Figure 1 describes the interaction process of the user with the tool, which happens via Command Prompt. The step-by-step of this interaction between the user and the tool is as follows:

(1) The user sends an input, which consists in a brief description of the bug found;
(2) The system assigns the input for one of the T5 specific language models after choosing the most adequate model based on the type of issue;
(3) The version of the T5 language model chosen by the system processes the user's input;
(4) The user receives a complete bug report, after being generated by the model.

In the following section we will discuss details regarding the planning of this experiment, based on the system we described in this section.

4 Study Design

4.1 Planning

4.1.1 Goal and Research Question. We defined the goal of this study using the goal definition template of the Goal Question Metric (GQM) [2] paradigm as follows:

"Analyze **the usage of an LLM by practitioners** for the purpose of **characterizing** with respect to their **perceptions of reporting bugs with LLM** from the point of view of the **researchers**

[4] https://www.python.org/
[5] https://www.java.com/

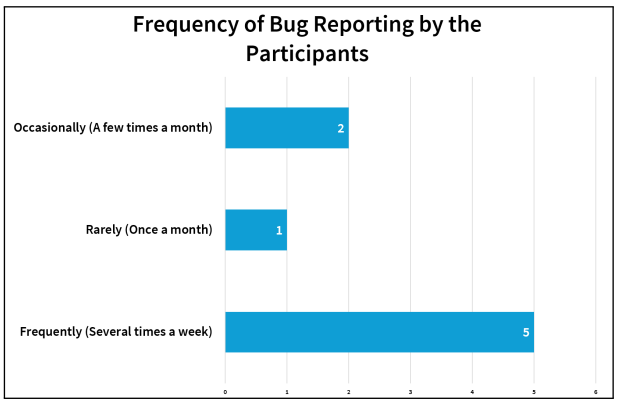

Figure 2: Frequency of bug reports written by the participants

in the context of **software testing team which employed LLM to generate bug reports during software testing execution**".

To investigate this goal, we formulate the following research question (RQ):

- **RQ1**: How much does LLM generate valid responses when reporting a bug?
- **RQ2**: How does a test team perceive the insertion of LLM to generate bug reports?

4.1.2 Context Description. This experiment was performed within a test team that acts in the software development of mobile devices in a software institute. The team is responsible for test execution, and if any problems are found during testing, the test team member responsible for the test must report the bug to the development team.

4.1.3 Selection of Participants. For this experiment, participants are test software professionals responsible for, among other activities, performing functional tests, as well as analysing and reporting bugs. We selected 8 members of the test team that report bugs, with different levels of experience and frequency of writing bug reports, for the experiment using non-probability convenience sampling [23]. The frequency of bug reports written by the participants is shown in Figure 2.

4.1.4 Instrumentation. We planned and revised all instruments to ensure that they were adequate for use in the experiment. This way, we performed a pilot experiment with 1 member of the test team, aiming to understand the quality of our materials. The participant selected for the pilot experiment was part of the population of testers from the institute, and this way we obtained feedback regarding the clarity of the instruments before their application in the main experiment.

This way, we produced the following instruments:

- **Informed Consent Form (ICF)**: A form received by the participants in which they agreed to participate in the experiment, thus maintaining the technical aspects of the research.
- **Presentation**: A presentation in which the researchers presented to the participants the goal of the experiment and how it would be conducted.
- **Bugs Sheet**: A sheet containing 5 bug descriptions and prompts for each bug, which would be used by the participants to register the LLM-based system's outputs and their perceptions regarding the validity of the output.
- **Post-Experiment Survey**: An online survey composed of 15 open questions, with the goal of characterizing the participants and obtain perceptions about the use of LLMs to report bugs.

4.1.5 Data Collection. The participants were selected as described in Section 4.1.3, and the experiment was conducted in a meeting room. The experiment was performed with 4 groups, each one composed of a pair of participants. In total, the experiment lasted for about an hour for each pair. The decision of conducting the experiment with pairs was motivated by the test team's project decisions, as their demands did not allow for the 8 participants to participate simultaneously. This way, the same script was followed with each one of the pairs, to ensure equity in the execution of the experiment.

The instruments described in Section 4.1.4 were used for collecting data: the bugs sheet was used to evaluate the system's outputs, and the post-experiment survey was used to collect feedback from the participants.

4.2 Execution

4.2.1 Preparation of the Bugs. Before starting the experiment with the participants, we selected the 5 types of bugs most reported by the test team. The goal of this selection was to measure the quality of the most frequently reported bugs, and thus receive feedback from the participants. Due to confidentiality policies of the institute, the bugs will be described in the following manner:

- **Bug A**: It is a bug in which it is not possible to perform adjustments in the network settings of the test device;
- **Bug B**: It is a bug in which an app is not functioning properly in the test device;
- **Bug C**: It is a bug in which there is a failure during the execution of a sequence of commands that perform validations in the test device;
- **Bug D**: It is a security bug in the test device;
- **Bug E**: It is a bug in which default settings are incorrectly set in the test device.

4.2.2 Conduction of the Experiment. The experiment was conducted in the following manner:

(1) **Presentation of the Experiment**: This first step lasted around 5 minutes, during which a slideshow was used to explain the script of the experiment.
(2) **Execution of the Experiment**: This step lasted 25 minutes, and participants performed the following tasks:
   - Verification of the type of bug in the bugs sheet;
   - Elaboration of the prompt to send to the system: Each participant was afforded the autonomy to generate their own prompt, and we refrained from intervening in this process;
   - Requesting the system to generate the bug report;
   - Evaluation of the output given by the system, where the participants could choose from the following options:

- "*Valid*", i.e., the participant agrees that the output generated by the LLM-based system is correct and it is according to the expected if the bug report were written by a human;
- "*Partially Valid*", i.e., the participant agrees that the output possesses valid structure, however there is still incorrect or missing information that needs to be corrected by a human;
- "*Invalid*", i.e., the participant states that the bug report is far from the expected result and contains wrong or missing information, and a human should fix all the bug report generated.

In all, 5 bug descriptions were used, and for each of them participants had to follow these same steps.

(3) **Post-Experiment Evaluation**: Lastly, after performing the experiment, there was a last step for collecting feedback regarding the use of LLMs for reporting bugs. This step lasted 30 minutes, in which the participant answered an online survey.

4.2.3 Analysis Procedures. We conducted the data analysis through 2 approaches:

(1) **Consolidation of Evaluations**: In this step, we summarized the evaluations by type of bug, presenting the statistical data corresponding to each type. The goal of this analysis was to answer RQ1;
(2) **Feedback Analysis**: In this step, we examined all the answers provided by the participants and assessed our findings. The goal of this analysis was to answer RQ2.

5 Findings

5.1 Analysis of LLM Responses

During the execution of the experiment, participants had to evaluate the output given by the system for each of the bugs that they send as input. The participants could choose from the following options: "valid", "partially valid", and "invalid". This way, the analysis presented here aims to answer our research question RQ1. Figure 3 displays the consolidation of all evaluations provided by the participants, as described below:

- **Bug A**: 62.5% of participants (5 participants) considered the output valid, while 25% (2 participants) considered it partially valid, and 12.5% considered it invalid (1 participant);
- **Bug B**: 87.5% of participants (7 participants) considered the output valid, and 12.5% participant considered it invalid (1 participant);
- **Bug C**: 62.5% of participants (5 participants) considered the output valid, while 25% (2 participants) considered it partially valid, and 12.5% of participants (1 participant) considered it invalid.
- **Bug D**: 50% of participants (4 participants) considered the output valid, while 25% (2 participants) considered it partially valid, and 25% of participants (2 participants) considered it invalid.
- **Bug E**: 25% of participants (2 participants) considered the output valid, while 25% (2 participants) considered it partially valid, and 50% of participants (4 participants) considered it invalid.

The presence of evaluations considered partially valid suggests that participants considered the output valid, but it still needs adjustments, as some of the information provided by the LLM was imprecise. In regards to the evaluations considered invalid, participants highlighted an important point: the answer generated by the LLM did not correspond to what was expected and contained incorrect information.

Types of bugs A, B and C achieved the best results, with evaluation percentages for valid and partially valid outputs above 87.5%, that is, at least 7 participants considered the output partially valid, confirming the efficacy of the LLM for reporting these types of bugs. In contrast, bugs D and E achieved a higher number of invalid evaluations, especially bug E, which 50% of participants considered the outputs as invalid, indicating the need for adjustments in the model to generate this kind of bug with more precision for the tester.

In all cases there were invalid evaluations, which indicates that while the LLMs generate valid outputs most times, there are still cases in which the LLMs produce incorrect answers. For this reason, human validation is still necessary, making the testers key for the validity of bug reports.

5.2 Analysis of Participants Perceptions

After performing the experiment, participants were asked to provide feedback about the use of LLM for reporting bugs. Thus, the analysis seeks to answer RQ2, highlighting how the test team perceived the use of LLM for reporting bugs.

5.2.1 Experience of Use. The feedback about the use of LLM as a tool for reporting bugs is positive. Participants considered it easy to use and intuitive, with most of the generated outputs being valid and needing but a few adjustments, as stated by P6: "*It was a pleasant and intuitive experience. I did not find hurdles to use it*" and as P1 complemented: "*Needs a few adjustments that do not impact the quality displayed*" The system was able to write precisely about common bugs that the participants report, though improvements in the description of the reproduction steps are still necessary. Despite the overall experience having been positively evaluated, some participants observed that the system could benefit from better refinement, particularly in the distinction among different kinds of bugs and among steps from different test cases, as noted by P4: "*The LLM is easy to use... however it might still be necessary to use a few examples for training with common words for different kinds of bugs... there was a mixture of steps from different test cases*" Despite the improvement points, the use of LLM for reporting bugs was considered a pleasant experience for its potential and ease of use for maintaining standard practices for bug reports.

Finding #1: The use of LLM as a tool for reporting bugs was positively received by the test team, being considered easy to use and intuitive.

5.2.2 Quality of the LLM's Outputs. The evaluation of the quality of the outputs generated by the tool is considered positive. Participants considered the outputs satisfactory, good and useful to optimize

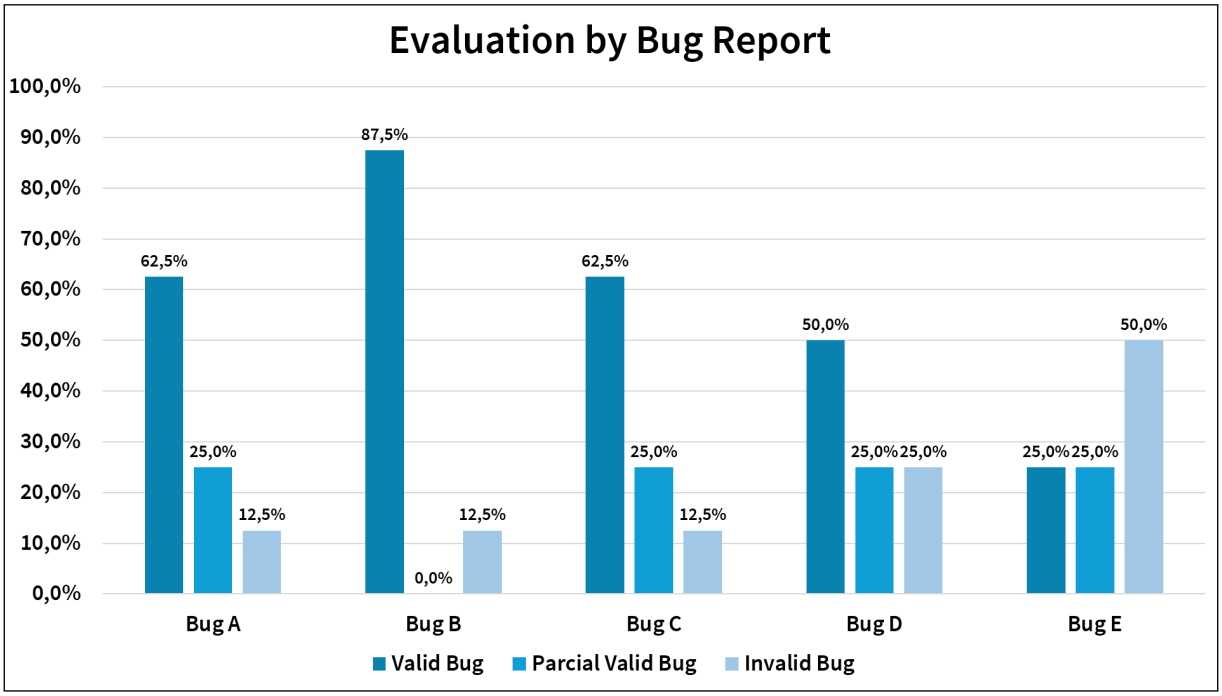

Figure 3: Evaluation of the system's output by type of report

the process of reporting bugs. The system provides an adequate initial base, preparing the text for a bug that might occur during tests. However, a few participants denote the need for adjustments and validations in certain cases, depending on the complexity of the bug. Participants also believed that the quality of outputs can improve with more specific and detailed information, as described by P5: "*They were good quality outputs. I believe the outputs will become better when we provide a higher amount of information*" and P6: "*In most cases is satisfactory and assertive. In a few cases, a few adjustments would be necessary to achieve a better result*". The tool is seen as promising, with potential for continuous improvement.

Finding #2: While a LLM can provide valid outputs, it is still necessary to perform adjustments and validations before reporting a bug. Human participation is still key for this process.

5.2.3 Intended Use of LLM Response for Reporting Bugs. Using LLMs to generate bug reports makes the process more dynamic and efficient, reducing the time required to report bugs compared to manually filling in fields, as mentioned by P1: "*Using an LLM tool to generate bug reports makes the process more dynamic... since currently it is necessary to fill in several fields because the current tool is static*". Participants contributed by stating that the generated responses generally contain the main correct information, although they may require minor adjustments or the addition of specific details. Most participants agreed that they would use generated response to report a bug, highlighting benefits such as speed, reduction of translation errors and ambiguity, and the ability to follow an appropriate step-by-step structure. However, participants also emphasized the importance of reviewing and adapting the information as needed, as stated by P4: "*Yes, to make the writing process faster, but being aware of the need to review and adapt the information according to the bug*".

Finding #3: The testing team considers LLM an effective tool for making the bug identification process more dynamic, optimizing time and reducing the need for manually filling in information.

5.2.4 Participants' Perceptions Facing Imprecise Responses. Participants also mentioned that they encountered some inaccurate or incomplete information provided by the LLM tool, but this did not significantly affect their overall perception of quality. They noted the need to review and complement the generated responses, identifying issues such as missing steps, inadequate descriptions, and the mixing of steps from different test cases, as mentioned by P3: "*...I found missing or poorly described steps. These occurrences made me pay more attention to what it writes and double-check*" and by P4: "*...There was a mix of some steps that are common to different test cases. This demonstrated the need to review the generated steps or include more details in the failure generation prompt*". Despite this, participants considered that the overall context was aligned with expectations, and that making minor adjustments after the failure was generated by the LLM would be more efficient than writing bug reports entirely manually. As highlighted by P1: "*It did not change my perception of the presented quality, as it was clear that even with some inaccurate or incomplete information, the overall context was in line with what is usually reported by the team. By making these*

manual adjustments, the user would spend less time than registering a bug in the current format used by the test team.". Participants also acknowledged the importance of carefully reviewing the responses before using them in a bug report.

Finding #4: Although an LLM generates some responses considered invalid by the participants, they recognize the importance of the human role in reviewing the responses before using them in a bug report.

5.2.5 Benefits of Using LLM. Participants highlighted several benefits associated with using LLM as a tool for automatic bug report generation, including the dynamism in the fields used, automatic filling based on test cases, the reduction in the time to record failures and in the occurrence of incorrect fields, in addition to maintaining the quality of the reports. They also emphasized the optimization of time, since the tool correctly describes most of the steps, allowing users to only increment and fill in gaps, as described by P3: "*Time optimized mainly, since it already describes most of the steps correctly, the idea of not having to write a bug report completely, but rather just increment and fill in gaps of what it didn't write is very promising*" and complemented by P5: "*Time gain when creating an issue report, especially in simpler and more straightforward issues...*". Participants also mentioned that the use of LLM is effective in generating bugs for more static cases (without major user-dependent steps), writing key test steps. Other positive points include the prevention of translation errors, the fast generation of complete structures with simple prompts, and the maintenance of the bug opening pattern, as mentioned by P7: "*Just with a simple prompt, a complete structure is made in a short time, something that would take time if done manually*" and P8: "*...The following of the bug opening pattern, the speed of template generation and ...the assertiveness of the information returned*".

Finding #5: LLM is a tool that describes most of the steps of a bug report, and the testing team just adds to it and fills in the missing gaps in a bug report.

5.2.6 Challenges Faced during LLM Usage. Participants also noted some negative aspects of using LLM as a tool for automatic bug report generation, including: missing important steps in the bugs, lack of clarity in some bug reports, confusion between similar bugs, unnecessary repetition of information, issues with preconditions in some bug reports, incorrect steps, need for additional information to better specify bugs, unexpected or unnecessary responses, and one occurrence of information generation with the ending compromised by random data. As detailed by P7:"*The main negative aspect was that one response came completely outside of what was expected, and... one response came with unnecessary information, which could completely change the outcome when reporting the bug*".

Finding #6: The testing team is capable of identifying problems resulting from failures generated by the LLM, demonstrating a critical view by recognizing that not all responses provided by the LLM are correct.

5.2.7 Perception of LLM Improvements and Contributions. Participants also indicated that using LLM as an automatic fault generation tool in the testing team will provide significant improvements, including time optimization, an increase in the amount of faults recorded in a short period, agility in the writing and reporting process, improvement in the quality of reports (especially in periods of high demand), greater accuracy in writing faults as LLM learns from the faults reported, a reduction in the time spent filling in information, more options for reporting faults and greater accuracy in opening faults. As P2 pointed out: "*Time optimization. More faults can be recorded in a shorter period of time*", detailed by P5: "*It will improve the agility of the reporting process and also the quality of the report, because at times of high demand the tester is overloaded, so they have to do this process in a more hurried way, which reduces the quality*" and P8: "*Agility and precision in opening faults*". The participants believed that the use of LLM will bring about these improvements, resulting in a more efficient process, allowing more projects to be dealt with in less time.

Finding #7: The use of LLM provides better quality bug reports.

5.2.8 Risks Observed. Participants also mentioned that using a tool to generate bug reports automatically can present some risks, mainly related to reducing the perception of incorrect writing by testers. The participants believe that it is necessary for testers to carefully review the content generated, as LLM can be confused in more complex or specific cases, generating unnecessary repetitions or incorrect information. As P2 pointed out: "*...We must look at a fault before it is recorded. We need to be attentive and analytical*". Thus, validation and adjustments by the tester are essential to avoid undue or incomplete faults being recorded. Although the participants considered the LLM to be a useful tool, especially for basic faults, they believed that it should not be assumed that the answers will always be correct, and that it is crucial to check that the result obtained corresponds to what is expected before using it in a report, as P4 emphasized: "*Validation by the tester is necessary because, in certain cases, the steps can be confused by the LLM or unnecessary repetitions can be generated*".

Finding #8: The use of LLM can entail risks, such as the possibility of the tester not identifying the incorrect writing of a bug report. However, this underscores the importance of the tester's fundamental role in double checking.

6 Threats and Limitations of the Study

This study has some threats and limitations:

- This study was carried out in a Software Institute with a specific development context, which adopts confidential policies to guide its operations. The research considered these particularities of the organizational environment, and it is therefore not possible to generalize the results to other institutes or test teams.
- The model developed in this study was trained with data specific to the bugs observed in the test team in question. Although this approach is effective for the particular context

analyzed, it has significant limitations in terms of generalization. The specificity of the training data restricts the model's applicability to other teams at the institute, because the characteristics and patterns of bugs can vary considerably between different groups. Consequently, using this model in other organizational contexts within the institute may not produce accurate or reliable results, thus limiting its overall usefulness.

- The study in question chose to focus its analysis on five specific types of bugs, although the model was trained with other types of bug report. This decision was made to concentrate efforts on the most frequent and relevant bugs, allowing for a more in-depth and detailed investigation of these cases. By limiting the scope of the analysis, researchers were able to examine the characteristics, patterns, and impacts of the five selected bugs. However, it is important to recognize that this approach can also have limitations, as it excludes the consideration of other types of bugs that, although less common, could provide valuable information about the model's performance and vulnerabilities in more diverse or unusual scenarios.
- The study sample consisted of eight participants, selected on the basis of the experience of the test team members to ensure that the study participants had experience in the process of opening bugs, guaranteeing greater homogeneity of the data collected. The participants were part of a representative sample, as they were software testers who used the tool, despite their small number.

7 Conclusions

This paper discusses the application of LLM to bug reporting using the adjusted T5 model to generate reproduction steps for bugs reported by a testing team in the software industry. Specifically, an experiment was conducted with the participation of 8 test team members responsible for reporting bugs. The experiment involved the evaluation of the five most common bugs reported by the testing team (cf. 4.2.1), followed by an analysis that collected participants' feedback on the use of LLM for bug reporting.

The results indicate that bug report types A, B, and C were highly accepted by the participants, with over 87.5% valid or partially valid evaluations, demonstrating the effectiveness of LLM in reporting these bug types. Bugs D and E had more invalid evaluations, particularly bug E, with 50% of the responses considered invalid. The test team positively evaluated the use of LLM to report bugs, considering it easy and intuitive and capable of optimizing the bug identification process. However, the importance of the human role in reviewing and validating the answers generated by the LLM before using them in bug reports is stressed, recognizing both the benefits and potential risks associated with using this tool.

The application of LLM to bug reporting has yielded promising results. The experiment revealed high acceptance of the responses of the T5 model adjusted to generate bug reports. However, there are still challenges in reporting bugs via LLM, indicating that there is still a need for adjustments in LLM training to generate bug reports correctly. In addition, the feedback provided by the testing team showed that participants saw that using LLM could optimize the bug identification process, highlighting its ease of use. However, it is crucial to emphasize the need for human supervision in reviewing and validating the answers generated by the LLM before implementing it in bug reporting. In future work, we intend to investigate other LLM techniques, such as RAG [9] to improve the quality of the responses generated by LLM.

ACKNOWLEDGMENTS

This paper is a result of the Research, Development & Innovation Project (ASTRO) performed at Sidia Institute of Science and Technology sponsored by Samsung Eletrônica da Amazônia Ltda., using resources under terms of Federal Law No. 8.387/1991, by having its disclosure and publicity in accordance with art. 39 of Decree No. 10.521/2020. The authors extend their gratitude to the Software Engineering Laboratory for its contribution to the LLM infrastructure.

REFERENCES

[1] Jagrit Acharya and Gouri Ginde. 2025. Can We Enhance Bug Report Quality Using LLMs?: An Empirical Study of LLM-Based Bug Report Generation. arXiv:2504.18804 [cs.SE] https://arxiv.org/abs/2504.18804

[2] Victor R Basili. 1994. Goal, question, metric paradigm. *Encyclopedia of software engineering* 1 (1994), 528–532.

[3] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiß, Rahul Premraj, and Thomas Zimmermann. 2007. Quality of bug reports in eclipse. In *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*. 21–25.

[4] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 308–318.

[5] Oscar Chaparro. 2017. Improving Bug Reporting, Duplicate Detection, and Localization. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 421–424. doi:10.1109/ICSE-C.2017.27

[6] Lennon Chaves, Flávia Oliveira, and Leonardo Tiago. 2024. Automating Issue Reporting in Software Testing: Lessons Learned from Using the Template Generator Tool. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering* (Porto de Galinhas, Brazil) *(FSE 2024)*. Association for Computing Machinery, New York, NY, USA, 278–282. doi:10.1145/3663529.3663847

[7] Lennon Chaves, Flavia Oliveira, and Leonardo Tiago. 2024. Enhancing Automated Tools: Reporting Bugs with Bug Builder. In *Anais do IX Simpósio Brasileiro de Testes de Software Sistemático e Automatizado* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 80–82. doi:10.5753/sast.2024.3674

[8] Sidong Feng and Chunyang Chen. 2024. Prompting is all you need: Automated android bug replay with large language models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[9] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] https://arxiv.org/abs/2312.10997

[10] Enyo José Tavares Gonçalves. 2010. Importância de Testes Sistêmicos para a Qualidade do Software. *Revista Tecnologia* 31, 1 (2010), 29–38.

[11] Davi Gonzaga, Leonardo Tiago, Ana Silva, Flávia Oliveira, and Lennon Chaves. 2025. Improving Bug Reporting by Fine-Tuning the T5 Model: An Evaluation in a Software Industry. In *Anais do X Simpósio Brasileiro de Testes de Software Sistemático e Automatizado* (Recife/PE). SBC, Porto Alegre, RS, Brasil, 141–143. doi:10.5753/sast.2025.13591

[12] Jueun Heo, Gibeom Kwon, Changwon Kwak, and Seonah Lee. 2024. A Comparison of Pretrained Models for Classifying Issue Reports. *IEEE Access* 12 (2024), 79568–79584. doi:10.1109/ACCESS.2024.3408688

[13] Abhishek Kumar, Sonia Haiduc, Partha Pratim Das, and Partha Pratim Chakrabarti. 2024. LLMs as Evaluators: A Novel Approach to Evaluate Bug Report Summarization. arXiv:2409.00630 [cs.SE] https://arxiv.org/abs/2409.00630

[14] Christopher D. Manning. 2022. Human Language Understanding & Reasoning. *Daedalus* 151, 2 (05 2022), 127–138. doi:10.1162/daed_a_01905 arXiv:https://direct.mit.edu/daed/article-pdf/151/2/127/2060607/daed_a_01905.pdf

[15] Jesse G Meyer, Ryan J Urbanowicz, Patrick CN Martin, Karen O'Connor, Ruowang Li, Pei-Chen Peng, Tiffani J Bright, Nicholas Tatonetti, Kyoung Jae Won, Graciela Gonzalez-Hernandez, et al. 2023. ChatGPT and large language models in academia: opportunities and challenges. *BioData mining* 16, 1 (2023), 20.

[16] Wendkûuni C Ouédraogo, Kader Kaboré, Haoye Tian, Yewei Song, Anil Koyuncu, Jacques Klein, David Lo, and Tegawendé F Bissyandé. 2024. Large-scale, Independent and Comprehensive study of the power of LLMs for test case generation. *arXiv preprint arXiv:2407.00225* (2024).

[17] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv:1910.10683 [cs.LG] https://arxiv.org/abs/1910.10683

[18] Betim Sherifi, Khaled Slhoub, and Fitzroy Nembhard. 2024. The Potential of LLMs in Automating Software Testing: From Generation to Reporting. arXiv:2501.00217 [cs.SE] https://arxiv.org/abs/2501.00217

[19] Yang Song. 2024. *Automated Bug Report Management to Enhance Software Development.* The College of William and Mary.

[20] Yang Song, Junayed Mahmud, Nadeeshan De Silva, Ying Zhou, Oscar Chaparro, Kevin Moran, Andrian Marcus, and Denys Poshyvanyk. 2023. Burt: A chatbot for interactive bug reporting. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion).* IEEE, 170–174.

[21] Shenglong Tan, Shenghong Hu, and Lihong Chen. 2010. A Framework of Bug Reporting System Based on Keywords Extraction and Auction Algorithm. In *2010 Fifth Annual ChinaGrid Conference.* 281–284. doi:10.1109/ChinaGrid.2010.13

[22] KS Thant and HHK Tin. 2023. The impact of manual and automatic testing on software testing efficiency and effectiveness. *Indian journal of science and research* 3, 3 (2023), 88–93.

[23] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. 2012. *Experimentation in Software Engineering.* Springer Publishing Company, Incorporated.

[24] Xin Xia, David Lo, Ming Wen, Emad Shihab, and Bo Zhou. 2014. An empirical study of bug report field reassignment. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE).* 174–183. doi:10.1109/CSMR-WCRE.2014.6747167

[25] Yi Yao, Jun Wang, Yabai Hu, Lifeng Wang, Yi Zhou, Jack Chen, Xuming Gai, Zhenming Wang, and Wenjun Liu. 2024. BugBlitz-AI: An Intelligent QA Assistant. In *2024 IEEE 15th International Conference on Software Engineering and Service Science (ICSESS).* 57–63. doi:10.1109/ICSESS62520.2024.10719045