GUIDE: A Framework for Improving Functional Software Test Descriptions with Language Models

Mathis Ronzon
Alten Labs, Rennes, France
mathis.ronzon2@alten.com

Thierry Roger
Alten Labs, Rennes, France
thierry.roger@alten.com

Zoltan Miklos
Univ Rennes CNRS IRISA
Rennes, France
zoltan.miklos@irisa.fr

Annie Foret
Univ Rennes CNRS IRISA
Rennes, France
annie.foret@irisa.fr

Abstract

Functional software testing is essential to ensure that software meets user expectations. Our ambition is to enable business experts who have extensive domain knowledge but limited software engineering competences, to realize the functional software tests, through formulating test case descriptions in natural language. To meet this challenge, we propose a framework called GUIDE (Guided User-driven Interactive Description Enhancement), which leverages language models to improve functional software test descriptions written in natural language. Our framework implements an intermediate step based on a structured language (Gherkin[1]) that is a language widely used for software tests. We translate test descriptions written by the business expert to this language using language models. We automatically evaluate the quality of the description based on the generated Gherkin. When this quality is insufficient, GUIDE initiates an interactive and personalized assistance process, delivering targeted advice to help business experts enhance and improve their test case descriptions. We evaluated our approach through a case study based on test cases for a human resources management related software, written in French. We recorded a 26% decrease in the average number of descriptions required per test objective to reach the desired quality level thanks to the advice generated.

1 Introduction

Software testing plays a crucial role in the quality and longevity of IT applications. However, a persistent divide between developers and end-users often complicates this essential task. While developers master the code and technical specifics, the business expert is the one who know the functional requirements. Nevertheless, they do not always have the tools or the language to express their expectations that is precisely understandable by the developers. This dissonance can compromise test reliability and software quality.

The advent of language models, capable of generating code from natural language instructions, is a promising solution for test production (Tufano et al., 2021; Xie et al., 2023). However, this approach is still mainly accessible to people with solid programming expertise, thus excluding many business experts. The latter also encounter difficulties in interacting with language models: they often give up too early when faced with a lack of understanding of the model, or formulate erroneous expectations based on dynamics specific to human interaction (Zamfrescu-Pereira et al., 2023). Moreover, each individual writes natural language text differently, even when the objective is identical (Weigelt et al., 2020). In parallel, some recent benchmarks (Jimenez et al., 2024) now include expert-verified and human-annotated versions of problem descriptions, acknowledging that instructions written by non-expert users are often insufficient to fully capture the functional intent behind a request.

In light of these challenges, it becomes essential to explore novel approaches that qualify user input and provide actionable guidance to improve it before giving it to a code generation pipeline. To this end, we propose a novel framework that we call GUIDE (*Guided User-driven Interactive Description Enhancement*), that aims to guide business experts in improving the quality of functional software test descriptions. The main objective of GUIDE is to enhance the clarity and precision of natural language test descriptions. More specifically, these descriptions are produced by a business expert without any specific rules being imposed,

[1] https://cucumber.io/docs/gherkin/reference/

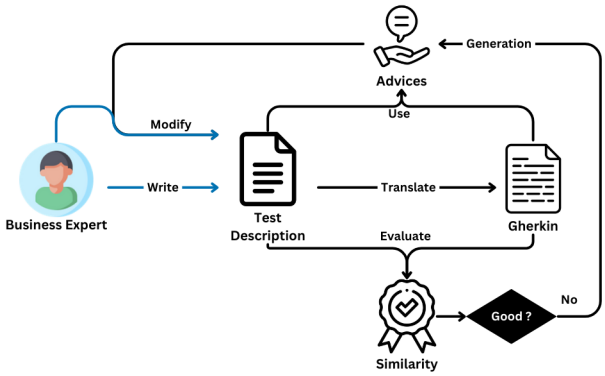

Figure 1: Overview of the GUIDE framework. In blue are the parts that the user is asked to perform, and in black those that are automated. Firstly, the user produces a description of a functional test, which is then translated into the Gherkin language using a language model. The quality of the test description is then assessed using this code. If the quality is judged to be insufficient, advice is generated based on the test description and the Gherkin code, enabling the user to modify the description himself.

and seek to explain how the system works according to a particular functionality. Our approach is based on three key components (Figure 1): (1) a quality criterion leveraging the automatic translation of descriptions into an intermediate representation language (Gherkin), (2) a classification system to evaluate the semantic similarity between the description and its translation, and (3) an interactive advice generation mechanism to guide users in refining their descriptions. A complete example of such a test description, its translation into Gherkin and their similarity according to our scale can be found in Table 1.

To evaluate the effectiveness and relevance of GUIDE, we seek to address the following research questions:

- **RQ1** : To what extends automatic translation of a test description into an intermediate language can constitute an appropriate quality criterion?
- **RQ2** : Can a small language model manage to understand the semantic similarities between a test description and its translation into an intermediate language?
- **RQ3** : Does an interactive process using automatic advice generation can help a user improve the quality of his description?

Thanks to the participation of 60 people, spread over three labelling campaigns and two production campaigns, we have been able to evaluate the effectiveness of GUIDE. The results show that 70% of the advice generated is considered relevant by users. In addition, we observed a measurable improvement in their ability to comply with the quality criteria, with a 26% reduction in the average number of descriptions needed per test objective to achieve the required level of quality.

The rest of the paper is organized as follows. We start by discussing the related works in Section 2. In Section 3, we introduce the three key components of our GUIDE framework in detail. Then, we discuss the practical implementation of GUIDE in a real-world experiment, where business experts interact with software under test conditions, that we describe in Section 4. We conclude the paper in Section 5.

2 Related Work

We review contributions related to large language models for code generation, techniques for refining ambiguous or incomplete user input, and methods for assessing the quality of natural language descriptions through text classification. We conclude by positioning our approach, GUIDE, in relation to these works.

2.1 LLM-based Code Generation

Large language models (LLMs) have emerged as a powerful tool for various code-related tasks, including program synthesis (Austin et al., 2021), bug fixing (Zubair et al., 2024) or program testing (Xiong et al., 2023). Through extensive pre-training, they recognize patterns, comprehend context, and generate coherent and contextually relevant code snippets.

In software testing, the use of natural language as an entry point remains limited. Approaches such as AthenaTest (Tufano et al., 2021) or A3Test (Alagarsamy et al., 2024) rely mainly on source code to generate tests, while ChatUniTest (Xie et al., 2023) uses a prompt composed mainly of code fragment.

2.2 Refining user input

The task of asking users to reformulate or modify their output is receiving increasing attention in the fields of information retrieval and dialogue systems. For example, Wang and Li (2021) propose a method based on question templates to help users clarify their requests. They use a genera-

Table 1: An example of a functional test description written by a non AI expert, translated into Gherkin code by an LLM and its similarity given by a human annotator using our similarity scale (Appendix A) (Example from our use case. The original description was written in French and can be found in the first row of Table 6)

Description	**Gherkin**	**Similarity Label**
Click on 'OPEN'. Check that the file selection window opens. Select a file with the '.csv' extension and check that the search appears in the software.	**Given** the software is open **When** I click on "OPEN" **Then** the file selection window opens **And** I select a file with the extension ".csv" **And** the search appears in the software	VERSIM

tion model based on a Transformer. Eberhart and McMillan (2022) propose a new method that uses a task extraction algorithm to identify aspects of the query and follows a rule-based procedure to generate questions.

In code generation, dealing with ambiguous user requirements has received more attention. Some pipeline such as QualityFlow (Hu et al., 2025) have integrated the evaluation of the quality of natural language requests and offers self-improvement mechanisms to reformulate instructions without the aid of the user. Other methods such as ClarifyGPT (Mu et al., 2023) or CodeClarQA (Li et al., 2022) questions the user to clarify ambiguities. In the case of ClarifyGPT, the ambiguity of a requirement is detected when several generations of code produced from the same instruction lead to different behaviours for the same input. As for CodeClarQA, it always asks questions, but has no system for assessing this ambiguity.

2.3 Text classification

To address the issue of ambiguous or incomplete test descriptions, recent research has focused on automated classification of textual quality. Traditional methods rely on logical rule-based systems that detect key phrases or syntactic patterns indicative of test completeness (Ormandjieva et al., 2007). Although effective in well-structured scenarios, these systems lack the flexibility to handle the linguistic diversity present in natural language inputs. Another method involves supervised learning based on standard metrics, representing the criteria that an expert takes into consideration when assessing the quality of requirements (Parra et al., 2015).

2.4 GUIDE positioning

We have seen that in current approaches to code generation, consideration of the quality of user input often comes after a long and costly process, once the initial generation has already been carried out. In contrast, GUIDE seeks to intervene upstream, validating the quality of the request as soon as it is created, to ensure that all the elements required for correct generation are present.

In addition, when ambiguity is detected, some approaches ask questions, sometimes of a technical nature, to fill in the missing information. GUIDE adopts a different strategy: it allows the user to modify the description directly, making implicit information explicit. This process is based on the generation of targeted advice, offered to the user in a non-binding way. In this way, the user retains control of their production, while being guided to improve it progressively.

3 The framework GUIDE

In this section, we will present our framework. In order to do so, we start by discussing quality criterion that we have chosen to discard the description. Then, we will look at the method used to assess automatically the quality of a test description using its translation into Gherkin code. Finally, we will present the method used to guide the user in the process of improving his production.

3.1 Quality criterion

Our approach is based on the use of a quality criterion to filter test descriptions according to their relevance. Our aim is to develop a method that retains only those test descriptions that contain all the elements necessary for a language model to both generate and verify the corresponding test. Thus,

the quality criterion aims to evaluate the ability of the language model to restore all the information contained inside the description.

Thus, to assess the language model's ability to understand the description, we use it in a simple task: information reorganisation. Specifically, we ask the language model to structure the information present in the test description. To do so, we use a intermediate language that meets this requirement perfectly: Gherkin. Gherkin is a human-understandable specification language used in the development method known as BDD (Behavior-Driven Development).

We will therefore base our quality criterion on the similarity level between the test description and its translation into Gherkin code by a language model. More specifically, it is based on a similarity scale detailed in Appendix A, which is composed of five labels:

- Similar : `COMPSIM`, `VERSIM`
- Different : `SOMSIM`, `VERDIFF`, `COMPDIFF`

By aggregating these labels into two distinct groups, we define our quality criterion: a description is considered of good quality if it obtains a similarity label of `COMPSIM` or `VERSIM`. Event though our quality criterion does not take advantage of all the nuance offered by the five similarity labels, we have choose to retain this scale. This additional granularity proves invaluable during manual labelling campaign, as it enables human annotators to better express their perception of the similarity. This finer distinction encourages a more precise and nuanced assessment.

3.2 Supervised Learning Quality Assessment

Two main approaches can be used to classify a text according to a given label: logical methods, based on rules or feature extraction, and approaches based on language models. However, in our case, the test descriptions present a high degree of lexical and structural heterogeneity, in addition to a dense technical content. These characteristics make logical methods ineffective, as they are too rigid and not very adaptable to the variability of the data.

We therefore opted to use language models, which are better able to capture the subtleties of natural language, even in a technical context. Given that our Framework is likely to handle sensitive or confidential data, we have deliberately restricted our choice to compact models that can be run locally, without depending on remote services.

3.3 Interactive Improvement Process

GUIDE not only qualifies the description written by the user, but also provides guidance when it is not of good quality. This guidance is intended to suggest possible changes that the user could make. We have chosen to use this form because we want the user to have a choice of modification throughout the procedure.

To produce it, we will use the description and its associated Gherkin code and give it, using a predefined prompt, to a language model. In order not to introduce our own bias into this generation of advice, and the evaluation of the prompts used to generate them, we have decided to use an automatic optimisation methods to find the best prompt, in particular Beam Search (Pryzant et al., 2023). This method is based on a starting prompt, a scoring metric and a method for generating several variations of a prompt to explore the space of available prompts.

The basic prompt is structured in three main parts: two dedicated slots for inserting the test description and its translation into Gherkin; another slot used to insert context of the software under test to maximise the relevance of the advice generated; an explicit sentence tells the language model the expected objective as well as the constraints to be respected (clarity, consistency and respect for the context).

For the mutation prompt, we ask the language model to produce three variations of a prompt, while retaining the meaning and the three slots reserved for description, Gherkin and context. These variations aim to explore different formulations while maintaining the structure of the task. An example of a basic prompt and a mutation prompt can be found at Appendix C.

The score metric is based on a realistic approach, aimed to simulate the behaviour of a user using the advice generated by a prompt. Using a dataset of test descriptions classified as of bad quality, and the prompt submitted for evaluation, we will produce advice for each of them. This advice, combined with the initial description and the context of the software under test, is then provided to a language model which simulates a user by producing a new version of the description, incorporating the suggested recommendations. This new descriptions,

together with its translation into Gherkin, is submitted to the classifier for quality evaluation. We calculate the prompt score by measuring the percentage of descriptions that, after modifications, pass the quality criterion.

4 Software Testing by business expert : Case study

In order to evaluate GUIDE in a real-life application, we chose to involve users, not necessarily with a background in IT, in the task of writing test descriptions. To this end, we set up three complementary campaigns as illustrated in Figure 2:

- A labelling campaign which has the objective to evaluate the relevance of `Gherkin` productions automatically generated by a language model.
- A production campaign which aims to observe how users behave when writing descriptions, measuring in particular their ability to produce content in line with our quality criterion.
- An advice campaign that seeks to assess user satisfaction with the use of advice produced by our method.

For the sake of simplicity, we decided to ask for the descriptions to be produced in French for all the campaigns. This allowed us to have more people available to take part in the campaigns.

4.1 Software under test

The software to be tested is called "Esco Explorer". It is a tool for displaying a graph in the form of an Acyclic Guided Graph (AGG) based on the occupations/skills given by the Esco ontology (Appendix B). Esco is a European classification of skills, competencies, qualifications and professions. The system identifies and categorizes skills, competences, qualifications and occupations relevant to the EU labor market, education and training, in 25 European languages. The system provides occupational profiles showing the relationships between occupations, skills, competences and qualifications; it functions like a dictionary.

During software development, a test plan consisting of 68 tests divided into 9 categories was produced. We used this same test breakdown for the rest of this section, enabling us to indicate a category and a precise test goal to the user, to help them write their description. An example of a category and its associated test purpose is: Category – Node Information, with the Test purpose – Display an optional job for a skill.

4.2 Campaigns

Labelling Campaign

Prior to the various experiments, five users were asked to write one natural language description per test purpose, resulting in five complete test plans. Based on these plans, we generated a manual labeling campaign aimed at assessing the quality of the descriptions via their correspondence with an automatically generated `Gherkin` code.

In concrete terms, each annotator was assigned a test plan, in which he or she had to select a description, consult the corresponding `Gherkin` code, generated by a language model, and then evaluate the similarity between the two elements. This evaluation was carried out using our similarity scale.

Production Campaign

During a second campaign, users will have to write test description for each test purpose themselves. For each test purpose, users will have to produce a test description then label the similarity between the description and its translation in Gherkin. If their description doesn't respect our quality criterion, they will have to modify their description and redo the labeling process.

In order to avoid blocking users when faced with cases they consider too complex, we have left open the possibility of changing the test to be described even when the quality criterion has not been met. However, to guarantee a minimum of reformulation effort, each user was required to propose at least two attempts to improve his initial description before having the possibility to abandon and do a new test.

Advice Campaign

Using the same protocol as the production campaign, this time we decided to add the tips generated from the prompt designed in Section 3.3. More specifically, when the user indicates that their description is not of sufficient quality, we offer them the advice generated from their description and the Gherkin. The user can then take these tips into account, or not use them if they don't find them interesting. This is indicated by two labels.

Gherkin Generation

In the interests of data governance and in order to guarantee local execution without dependency on

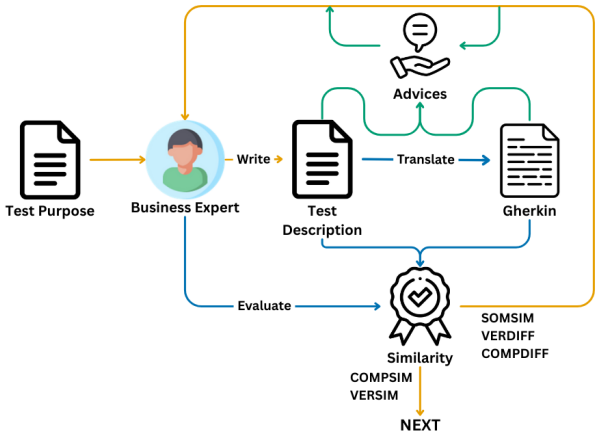

Figure 2: Overview of the different campaigns. The blue color represents the labelling campaign, where users assess the similarity between a previously written description and its Gherkin translation. The orange indicates the production campaign, in which users write a description based on a given test purpose, evaluate its quality, and revise it if necessary. Finally, the green corresponds to the advice campaign, where users receive guidance generated from their initial description and its Gherkin translation to help them improve their text.

external services, we opted to use the Mistral V0.3-7b language model (Jiang et al., 2023), quantised in 4 bits. The prompt used to generate Gherkin was manually optimised using a set of descriptions considered to be of good quality.

4.3 GUIDE Implementation

4.3.1 Quality Classifier

In order to evaluate several possible classifiers based on language models, we have chosen to use CamemBERT v2 (Antoun et al., 2024) and SomlLM2-135M (Allal et al., 2025) as a basis. CamemBERT v2 is a robust and high-performance reference for automatic language processing tasks in French. SmolLM2-135M is a more recent, lightweight model, that is recognised for its good general capabilities despite its small size. This choice makes it possible to combine confidentiality, linguistic performance and operational efficiency.

Since SmolLM2 is trained exclusively on English data, we explored its viability by automatically translating our dataset into English using the opus-mt-fr-en (Tiedemann et al., 2023; Tiedemann and Thottingal, 2020) model. This model allows efficient conversion of descriptions and Gherkins written in French into English, ensuring consistent basis for training.

To train our classification models, we used the data collected from two campaigns described in Section 4.2: the labelling campaign and the production campaign. This resulted in a dataset of 1522 test description / Gherkin code pairs. Each pair was transformed into a single classifier input using explicit keywords (Description:, Gherkin:) and a [SEP] separator token.

Based on our quality criterion, we converted the original similarity labels into binary quality labels: 885 instances were labeled as `SIMILAR` (indicating sufficient quality), and 637 as `DIFFERENT`. The dataset was split into a 70/30 ratio for training and validation. All models were trained with 3 epochs and a batch size of 8.

To complement this training set, we also constructed a separate test set of 68 description/Gherkin pairs. These descriptions were written by a single user not involved in the previous datasets, and each pair was annotated five times. The final label for each instance was computed as an average label, using the method described in Section 4.4.1.

Table 2: Performance of classification models in training and evaluation

Model	Training Accuracy	Evaluation Accuracy	Test Accuracy
CBERT-Mix	81.78	74.55	60.29
Smol-FR	98.97	73.52	70.59
Smol-EN	98.97	72.23	58.82

The training results for the different models can be found inside Table 2. The models based on SmolLM2 show better accuracy during training, but this superiority is not reflected on the evaluation set, where the performance is similar to that of the models based on CamemBERT. Furthermore, translating the data into English did not bring any significant improvement in terms of accuracy.

Of the models evaluated, only one managed to maintain good accuracy over the test set: Smol-FR. The other two models showed a significant drop in performance, indicating a more limited ability to generalise. As a result, we have chosen Smol-FR for the rest of our experiments.

4.3.2 Advice Prompt

Using this classifier, we were able to launch the BeamSearch algorithm to produce the prompt used to generate the advice. We retrieved the first paragraph of Section 4.1 to be used as the context of the application under test inside the advice prompt.

As for the description dataset required for the score metric, we selected the descriptions from the production campaign (presented in the in Section 4.2), which were identified as being of insufficient quality by the classifier as well as the user who produced the description.

In total, our method evaluated 45 prompts each of which was evaluated with 218 data items. The selected prompt (found in Listing 3) received a score of 0.79.

4.4 Data Analysis

In this section, we will analyse the data from the three campaigns presented above. We will seek to answer the various research questions we had, as well as assessing the usefulness of GUIDE.

4.4.1 Gherkin Generation ability

Mean Label

A total of 21 people took part in the labelling campaign. Of the five test plans proposed, three were annotated by five people, while the other two were annotated by three people. Since several annotators evaluated the same pairs of data (test description and `Gherkin` code generated by an LLM), it is necessary to assign a consensus similarity label of each piece of data. To achieve this, we adopt a majority voting approach.

Using the decomposition of the five similarity label according to our quality criterion, we will look at the group with the most labels. Inside this majority group, if one label stand out with a clear majority, it is selected. Otherwise, we proceed to average the labels to select the most representative.

Model capability

Looking at the distribution of average labels obtained for each test plan (Table 3), we can see that some users, with no prior knowledge of the Gherkin language or the quality criterion used, manage to produce descriptions that directly satisfy this criterion. However, this success is not homogeneous: other test plans present initial descriptions whose quality is insufficient according to our quality criterion.

Despite these disparities, one encouraging point stands out: no test plan is completely misunderstood by the language model. This illustrates the robust ability of the selected model (MistralV0.3-7b) to interpret even descriptions from non-expert authors, and to produce usable Gherkin translations.

Table 3: Distribution of the similarity label depending of the Test plan.

Test Plan	COMPSIM	VERSIM	SOMSIM	VERDIFF	COMPDIFF
1	12	11	15	11	19
2	43	16	5	3	1
3	4	18	22	19	5
4	7	28	15	12	6
5	26	37	2	3	—

That said, a qualitative analysis of the comments left by annotators allows us to distinguish two main sources of error in Gherkin generation: problems of structure, linked to poor syntactic or logical organization of the generated code, and problems of ambiguity, due to an incomplete or poorly formulated initial description. This ambiguity is due to the annotator, who did not necessarily understand the test description correctly, as he pointed out in his commentary. It is therefore a semantic ambiguity, linked to imprecise wording or wording that is open to several interpretations in the initial description. Table 4 shows, for each test plan, the proportion of descriptions identified as having these two types of problem.

Table 4: Distribution of Structure and Ambiguity Errors in Low-Quality Descriptions per Test Plan (as Labeled by Annotators)

Test Plan	Structure (%)	Ambiguity (%)
1	90.2	58.6
2	76.3	88.9
3	89.2	73.0
4	90.7	65.8
5	55.2	51.8

We find that, in the majority of cases, failures to meet our quality criterion stem first and foremost from problems with the model's structuring of the Gherkin, with rates exceeding 90% in some shots. However, these structural errors are often exacerbated by poorly constructed initial descriptions, as shown by the high rate of ambiguity problems - reaching 88.9% in plan 9. This twofold observation highlights both the current limitations of the language model in correctly handling Gherkin's syntactic constraints, and the need to support users in improving the clarity and completeness of their descriptions.

4.4.2 User behaviour

Based on the results presented in Table 8, we observe that all users needed to revise their descriptions at least once, and in many cases several times, before reaching a level that matched our defined quality criterion. This reinforces the idea that generating a high-quality test description is not straightforward, especially for non-expert users.

However, the low number of abandons suggests that the effort required to improve a description is not perceived as excessive. In particular, only 36 abandons were recorded out of 680 attempts, indicating that most users were willing to iterate to reach the expected quality level.

Each user had to produce descriptions for 9 different test categories, in a fixed order that was identical for everyone. This enabled us to observe a potential progression. However, no clear trend emerged. We find this to be the case even when we split each categories into two equal halves (Table 7). No systematic improvement dynamic can be observed. So we don't need to take into account a history for each user in our GUIDE framework.

4.4.3 Advice capability

In total, 125 advice have been generated, with an overall satisfaction rate of 70%. This result indicated that the majority of users were positive about the usefulness of the advice provided.

The analysis also shows a reduction in the average number of descriptions per test: this drops from 1.75 (observed during the initial production campaign) to 1.29 during this campaign (Table 9). This reduction suggests that the advice makes it easier to achieve the quality criteria, thereby reducing the number of iterations required.

In addition, result in Table 5 reveal a marked difference between advice that is considered relevant and advice considered uninteresting. More specifically, advice perceived as useful is significantly more associated with improvements in the quality of the description. This trend suggests that the perceived quality of the advice has a direct influence on the user's ability to refine their description, thereby reinforcing the effectiveness of GUIDE's interactive process.

Table 5: Improvement of the similarity label depending on whether the advice has been deemed relevant by the user.

Improvement	Interesting	Not Interesting
Upgrade	51	11
Constant	34	24
Downgrade	2	3

5 Conclusion

This work introduced GUIDE (Guided User-driven Interactive Description Enhancement), a framework that improves the quality of test descriptions written by business experts through an interactive process. By leveraging Gherkin as an intermediate representation, GUIDE effectively assesses description quality and provides personalized advice for refinement, enabling non-technical users to produce clearer and more complete test scenarios.

Our experiments show that small language models like CamemBERT and SmolLM2 successfully identify semantic similarities between natural language descriptions and their Gherkin translations while maintaining data privacy through local processing. Additionally, the interactive advice mechanism reduces the number of attempts required to meet quality standards by 26%, highlighting its effectiveness in user-driven improvements.

While GUIDE has shown promise in improving the quality of business-driven test descriptions, several avenues for improvement remain open. Notably, we observed issues related to the structuring of Gherkin translations during the evaluation process. Despite its structured format, Gherkin generated by the translation step sometimes suffers from syntactic inconsistencies or incorrect formatting, which can hinder the subsequent classification and assessment. To address this limitation, it could be possible to use syntax-aware models that validate Gherkin structure during generation, or to apply post-processing corrections to ensure compliance with Gherkin's strict syntax.

Limitations

One of the core design choices of GUIDE is the use of small language models (CamemBERT and SmolLM2) to ensure local execution and respect for data privacy. While this choice enables on-premises deployment and reduces dependency on external cloud services, it also introduces a limitation in terms of generalization. Unlike larger pre-trained models (e.g., GPT-4, PaLM), smaller models require more task-specific fine-tuning to perform adequately. This additional training phase

can cause GUIDE to become more domain-specific, potentially limiting its effectiveness when exposed to new application contexts or unseen business-specific terminologies.

Moreover, GUIDE relies heavily on Gherkin as an intermediate representation to assess the quality of test descriptions. While Gherkin is well-structured and human-readable, it enforces a rigid format that may not capture more complex testing logic or non-linear interactions described by business experts.

References

Saranya Alagarsamy, Chakkrit Tantithamthavorn, and Aldeida Aleti. 2024. A3test: Assertion-augmented automated test case generation. *Information and Software Technology*, 176:107565.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. 2025. Smollm2: When smol goes big – data-centric training of a small language model.

Wissam Antoun, Francis Kulumba, Rian Touchent, Éric de la Clergerie, Benoît Sagot, and Djamé Seddah. 2024. Camembert 2.0: A smarter french language model aged to perfection.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models.

Zachary Eberhart and Collin McMillan. 2022. Generating clarifying questions for query refinement in source code search. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 140–151. IEEE.

Yaojie Hu, Qiang Zhou, Qihong Chen, Xiaopeng Li, Linbo Liu, Dejiao Zhang, Amit Kachroo, Talha Oz, and Omer Tripp. 2025. QualityFlow: An agentic workflow for program synthesis controlled by LLM quality checks.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Haau-Sing Li, Mohsen Mesgar, André FT Martins, and Iryna Gurevych. 2022. Python code generation by asking clarification questions. *arXiv preprint arXiv:2212.09885*.

Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binquan Zhang, Chenxue Wang, Shichao Liu, and Qing Wang. 2023. Clarifygpt: Empowering llm-based code generation with intention clarification. *arXiv preprint arXiv:2310.10996*.

Olga Ormandjieva, Ishrar Hussain, and Leila Kosseim. 2007. Toward a text classification system for the quality assessment of software requirements written in natural language. In *Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting*, pages 39–45.

Eugenio Parra, Christos Dimou, Juan Llorens, Valentín Moreno, and Anabel Fraga. 2015. A methodology for the classification of quality of requirements using machine learning techniques. *Information and Software Technology*, 67:180–195.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic Prompt Optimization with "Gradient Descent" and Beam Search.

Jörg Tiedemann, Mikko Aulamo, Daria Bakshandaeva, Michele Boggia, Stig-Arne Grönroos, Tommi Nieminen, Alessandro Raganato Yves Scherrer, Raul Vazquez, and Sami Virpioja. 2023. Democratizing neural machine translation with OPUS-MT. *Language Resources and Evaluation*, pages 713–755.

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.

Michele Tufano, Dawn Drain, Alexey Svyatkovskiy, Shao Kun Deng, and Neel Sundaresan. 2021. Unit Test Case Generation with Transformers and Focal Context.

Jian Wang and Wenjie Li. 2021. Template-guided clarifying question generation for web search clarification. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 3468–3472.

Sebastian Weigelt, Vanessa Steurer, and Walter F. Tichy. 2020. At your Command! An Empirical Study on How LaypersonsTeach Robots New Functions.

Zhuokui Xie, Yinghao Chen, Chen Zhi, Shuiguang Deng, and Jianwei Yin. 2023. ChatUniTest: a ChatGPT-based automated unit test generation tool.

Weimin Xiong, Yiwen Guo, and Hao Chen. 2023. The program testing ability of large language models for code. *arXiv preprint arXiv:2310.05727*.

J D Zamfrescu-Pereira, Richmond Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts.

Fida Zubair, Maryam Al-Hitmi, and Cagatay Catal. 2024. The use of large language models for program repair. *Computer Standards & Interfaces*, page 103951.

A Similarity Scale

- **Completely Similar** (`COMPSIM`): it is the same test and the same procedure expressed a little differently.
- **Very Similar** (`VERSIM`): it is the same test, but the operating procedures are different (one may be more detailed than the other, but that does not mean that the test described is different).it is the same test, but the operating procedures are slightly different.
- **Somewhat Similar** (`SOMSIM`): it is probably the same test (but I am not sure) and/or there are many differences in the operating methods used.
- **Very Different** (`VERDIFF`): it may be the same test and/or the operating procedures expressed have too many differences (but elements in common).
- **Completely Different** (`COMPDIFF`): it is not the same test and/or the operating procedures are completely different (no common elements).

B Esco Explorer

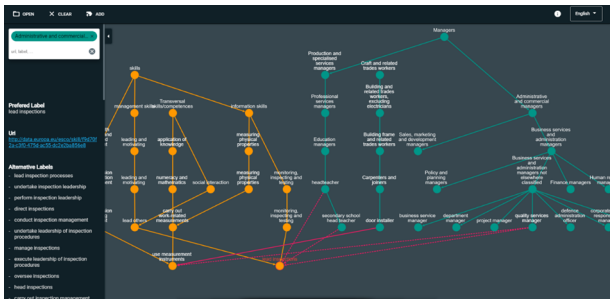

Figure 3: Screenshot of the software Esco Explorer

C Prompt for the Beam Search

All the prompts presented in this appendix were originally written in French.

Listing 1: The prompt used as the basis for the Beam Search.

```
The user wrote a test description, and an automatic
analysis identified the elements that were missing
or needed to be improved.
Based on the following information:
- Test description provided:
"%s"

- Gherkin:
"%s"

- Software context:
"%s"

Generate clear, actionable suggestions to help the
user improve their description. Your recommendations
 must be precise and adapted to the elements
detected as insufficient or missing. Respond only to
 suggestions and nothing else.
```

Listing 2: The prompt used to make the mutation during the Beam Search.

```
Generates three variations of the following
instruction, while retaining its semantic meaning.
Each variation must respect the following
constraints:

The message must remain clear and understandable.
The three %s markers must be retained, the first for
 the description, the second for the Gherkin and the
 third for the software.
The variations must reformulate the instruction
without changing its content.
Separate each variation with "---".

Reference instruction:
"%s"
```

Listing 3: The prompt considered to be the best during the Beam Search.

```
The user has drawn up a test description, and an
automatic analysis has highlighted elements to be
enhanced or corrected.

Based on the following information:
- Test description provided:
"%s"

- Gherkin:
"%s

- Software context:
"%s

Make concrete, achievable suggestions to help the
user improve his description. Your recommendations
must be explicit and adapted to the elements deemed
to be missing or insufficient. Answer only the
suggestions and nothing else.
```

D Data

Table 6: Examples illustrating the disparity of functional test descriptions produced by different users. These descriptions, taken from the production campaign, all aim to express the same test objective: 'File - Open a CSV file'. The similarity label was assigned by the user who wrote the description

Description	**Gherkin**	**Similarity Label**
Cliquer sur "OPEN". Vérifier que la fenêtre de choix des fichiers s'ouvre. Sélectionner un fichier avec l'extension ".csv" et vérifier que la recherche apparaît dans le logiciel.	**Given** le logiciel est ouvert **When** je clique sur "OPEN" **Then** la fenêtre de choix des fichiers s'ouvre **And** je sélectionne un fichier avec l'extension ".csv" **And** la recherche apparaît dans le logiciel	VERSIM
Dans ESCO Explorer, appuyer sur le bouton "OPEN" tout en haut à gauche. Une fois fait, chercher dans la liste un fichier avec comme type de fichier "Fichier CSV Microsoft Excel". Double cliquer sur le nom du fichier pour l'ouvrir. Si le fichier s'ouvre dans ESCO, on peut dire que le test est validé.	**Given** l'utilisateur est sur l'application ESCO. **When** l'utilisateur sélectionne un fichier CSV via le bouton "OPEN". **Then** le fichier CSV choisi devrait s'ouvrir dans ESCO.	COMPSIM
Cliquer sur le bouton 'OPEN' de la barre de menu d'EscoExplorer. Cliquer ensuite sur le fichier nommé 'cobolview.csv' dans la fenetre qui s'est ouverte puis cliquer sur le bouton 'ouvrir'. Le test est réussi si et seulement si le mot 'COBOL' apparaît dans la fenetre de recherche d'EscoExplorer.	**Given** EscoExplorer a été lancé **And** le langage sélectionné est 'English' **When** cliquer sur le bouton 'OPEN' **And** ouvrir le fichier 'cobolview.csv' **Then** 'COBOL' apparait dans la fenetre de recherche	SOMSIM

E Campaign Analysis

Table 7: Average number of productions per test category, dividing the test categories into two halves.

Test Category	1	2	3	4	5	6	7	8	9
First Half	2.97	1.5	2.6	1.45	1.8	1.52	1.62	1.63	1.5
Second Half	2.02	1.6	1.72	1.9	1.47	1.63	1.5	1.62	1.43

Table 8: User Behavior During the Production Campaign (Number of attempts to meet quality criterion)

User ID	Avg. Attempts	1 Attempt	2 Attempts	3 Attempts	3+ Attempts	Abandonment
0	1.71	42	12	7	4	3
1	1.15	59	8	1	–	–
2	3.07	18	18	12	20	–
3	1.91	32	19	12	5	–
4	2.31	21	11	10	1	25
5	1.79	35	18	8	5	2
6	1.29	53	12	2	1	–
7	1.38	51	13	2	2	–
8	1.44	45	16	6	–	1
9	1.41	46	12	5	–	5
Total	1.75	402	139	65	38	36

Table 9: User Behavior During the Advice Campaign (Number of attempts to meet quality criterion)

User ID	Avg. Attempts	1 Attempt	2 Attempts	3 Attempts	3+ Attempts	Abandonment
10	1.21	60	6	–	2	–
11	1.06	64	4	–	–	1
12	1.19	59	6	2	1	1
13	1.16	58	9	1	–	–
14	1.15	59	8	1	–	–
15	1.15	49	11	4	4	3
16	1.18	60	4	4	–	–
17	1.85	36	8	22	2	18
Total	1.29	445	46	34	9	23