From User Stories to Automated Scriptless Testing via Language Models: the AUTOLINK Project

Ashley T. van Can[1,†], Fernando Pastor Ricós[2,3,†], Tanja E.J. Vos[2,3,†] and Fabiano Dalpiaz[1,*,†]

[1]Utrecht University, Utrecht, the Netherlands
[2]Universitat Politècnica de València, Valencia, Spain
[3]Open Universiteit, Heerlen, the Netherlands

Abstract

Both requirements engineering and software testing are often regarded as tedious activities by practitioners, who tend to spend minimal effort on such tasks. Comprehensive specification documents have been replaced, in several contexts, by lightweight notations like user stories and acceptance criteria. Software tests are notoriously expensive to create and especially to maintain, leading to outdated test cases. In the AUTOLINK project, we are devising lightweight, (semi-)automated solutions that make use of NLP techniques to bridge the requirements engineering and software testing disciplines. Our NLP techniques extract a wide range of information from compact requirements representations, to then use it to automatically test software without the need of creating and maintaining test scripts. In this paper, we outline the project, describe the results we achieved so far, and sketch future directions for the project as well as for the broader research field.

Keywords

User Stories, Scriptless Testing, Requirements Engineering, Software Testing, Language Models, AUTOLINK

1. Introduction

Requirements engineering and software testing are disciplines of software engineering that are notoriously known for including tedious and repetitive activities [1, 2]. In particular, writing and maintaining high-quality artifacts, such as requirements and test cases, is time consuming and is not always a priority, especially in agile development settings where working software is prioritized over extensive documentation [3].

In requirements engineering, with the exception of highly regulated domains and safety-critical systems, we have witnessed a transition from comprehensive specification documents to lightweight, informal ways of documenting requirements like user stories stored in product backlogs [4, 5]. In software testing, it is well known that writing and maintaining tests is time consuming [6], and solutions have been devised toward automated scriptless testing, like the TESTAR tool [7].

The AUTOLINK project (Automated Unobtrusive Techniques for LINKing requirements and testing in agile software development) aims to tackle some of these challenges by proposing automated techniques, powered by natural language processing and machine learning, that integrate requirements engineering with software testing in agile practices. The key project pillar is that automation can help both disciplines, as long as we can identify effective ways to match requirements artifacts with software testing, in such a way that (i) requirements can inform the automated testing of a software system, and that (ii) testing information can be traced back to the requirements and support their verification.

AUTOLINK is funded by the Dutch Research Council and it is guided by two research groups (at Utrecht University and at the Open University) that collaborate with a user committee consisting of

Joint Proceedings of REFSQ-2026 Workshops, Doctoral Symposium, Posters & Tools Track, and Education and Training Track. Co-located with REFSQ 2026. Poznan, Poland, March 23-26, 2026

*Corresponding author.
†These authors contributed equally.
a.t.vancan@uu.nl (A. T. van Can); ferpasri@inf.upv.es (F. P. Ricós); Tanja.Vos@ou.nl (T. E.J. Vos); f.dalpiaz@uu.nl (F. Dalpiaz)
0009-0001-1190-8327 (A. T. van Can); 0000-0002-5790-193X (F. P. Ricós); 0000-0002-6003-9113 (T. E.J. Vos); 0000-0003-4480-3887 (F. Dalpiaz)

industrial partners: ING Bank, Mendix Technology, TestCompass, Newspark, AXINI, and B00. These partners facilitate the research by providing research data, enabling case studies, and by providing continuous feedback to the research teams.

The project is structured into four work packages: *WP1.* Automated techniques for tracing requirements downward; *WP2.* Automated scriptless testing of user stories; *WP3.* Integration and tooling; and *WP4.* Validation. In the rest of this project report, we focus on the first two work packages, as the latter two are supporting the former. We detail the currently achieved results in Section 2, then sketch ongoing and future directions in Section 3, and conclude in Section 4.

2. Current Results

We report on the results that have been published, with an emphasis on the use of NLP techniques on requirements-relevant artifacts. We include two sub-sections, one per work package.

2.1. Automated techniques for tracing requirements downward (WP1)

WP1 is concerned with the design and development of NLP-driven, unobtrusive techniques that create trace links in a downward fashion. By unobtrusive, we mean that these techniques should support the current way of working, rather than requiring or proposing alternative development methods. The overall aim of WP1 is that of matching widely adopted agile requirements artifacts, such as user stories and acceptance criteria, with the information about the system's UI that is extracted from automated scriptless testing via the TESTAR tool [7].

The results achieved so far relate to the automated extraction of information from requirements as expressed in product backlogs, without setting strong constraints on the format of such requirements. Although the original idea of the project was that of experimenting with classical NLP methods and machine & deep learning techniques, we quickly switched our focus to the various types of large language models (LLMs) due to their sudden availability and popularity.

Locating Requirements in Issue Tracking Systems. We started with an empirical investigation of how requirements are expressed in practice. To do so, we gathered a collection of product backlogs – in the form of issues extracted from issue tracking systems like JIRA – both from open source projects [8, 9] and from one of the larger industrial partners of the project: Mendix Technology. Through a manual analysis of a sample of 1,636 product backlog items [10], we found that (i) the labeling of backlog items is largely inconsistent, e.g., the label 'story' often misused to represent other categories; (ii) the most frequent requirements are user-oriented functional requirements; and (iii) more than 40% of the issues include multiple requirements, commonly following a refinement pattern where a user story is broken down into one or more tasks or acceptance criteria.

These results motivated us to devise and experiment with NLP approaches for the automated identification and classification of requirements in issues / backlog items [11]. In particular, we experimented with various LLMs, both encoder-only (BERT, RoBERTa) and decoder-only (Llama3-8B, Mistral-7B and ChatGPT with GPT-4) for two tasks: (i) determining whether a segment (i.e., a paragraph or a sentence) in an issue contains a requirement; and (ii) classifying a requirement segment according to the requirements type (user-oriented functional, system-oriented functional, non-functional) and granularity (low - like acceptance criteria, medium - like user stories, high - like epics).

We evaluated the encoder-only models, which (unlike decoder-only models) require training, via group k-fold cross-validation. For ChatGPT, we examined the effect of in-context learning by selecting examples from other datasets. Our experimentation showed that (i) encoder-only models significantly outperformed decoder-only models, with the best F1-score for encoder-only models: 0.62 (requirements class) and 0.88 (non-requirements class); for decoder-only models: 0.47 and 0.77; (ii) user-oriented functional requirements are the easiest to classify (F1-score of up to 0.81 for RoBERTa); (iii) identifying system-oriented functional requirements and non-functional requirements is challenging (best F1-score 0.40 and 0.54 for RoBERTa, respectively).

Extracting Domain Models from User Stories. In an attempt to extract relevant information from user stories that can be used for testing, we conducted research concerning the automated extraction of domain models from user stories, as the latter are the most common notation for expressing requirement in agile development [12]. Domain models can be used as a basis for testing in the sense that they provide an overview of the terms that are mentioned in the requirements, and these terms can then be matched to labels and names of UI components, for example.

We first constructed a ground truth that can be used to assess the effectiveness of automated solutions for the extraction of requirements [13]. We first compared classical NLP, specifically the Visual Narrator tool [14] that we developed almost a decade ago, against a machine learning solution (based on Random Forest) and ChatGPT (GPT-4). The results highlighted that none of the automated approaches could reach the performance of expert modelers. When comparing the automated treatments, we could not find a clear winner, with GPT-4 and machine learning achieving the best results in terms of recall and precision, respectively.

We then conducted additional experiments [15] with different prompting strategies and with the inclusion of Mistral0.3-7B; moreover, we compared the performance to that of novice modelers (undergraduate students). We found that machine learning and LLMs perform as well as novice modelers. Also, through a qualitative analysis of the false positives, we found that the various automated tools show different error profiles. For example, the Visual Narrator includes all roles as classes, while LLMs tend to include irrelevant implementation constructs.

On Energy Efficiency when using LLMs. The extensive experimentation with LLMs in the project, and the high computational cost that they require (especially compared to lightweight techniques like the Visual Narrator, which uses a classical NLP pipeline), made us initiate studies on the energy efficiency of the automated solutions we have been devising.

For the task of extracting domain models from user stories, we have conducted a comparative study between large and small language models [16]: GPT-o1 (large) was compared to Llama3-8B and Qwen-14B (small). Unlike GPT-o1, the small language models could be deployed locally on a desktop machine, thereby limiting their energy consumption and mitigating data confidentiality risks.

The results of our investigation showed that GPT-o1 outperformed the small language models in most cases (e.g., for class identification, the macro F1 score for GPT-o1 was 0.663, whereas Llama3-8B and Qwen-14B achieved 0.49 and 0.525), thereby highlighting the trade-off between energy efficiency and accuracy.

In many LLM experiments in the AUTOLINK project, we had to choose whether prompting all prediction instances and tasks to the LLM at once or in batches (e.g., in [11]). Since this consideration can affect performance or energy consumption, we decided to further investigate this trade-off. Specifically, we focused on the classical task [17] of classifying functional and non-functional requirements [18], and experimented with how many instances (requirements) should be given as input to an LLM for each prompt, i.e., the inference batch size parameter. Using an inference batch size of 64, for example, we can classify up to 64 requirements in a single shot. Our results showed that each language model exhibited a different pattern, and we could not determine a standard inference batch size that would always yield the best results. Yet, while larger batch sizes have somewhat lower accuracy, the degradation is not steep and it is compensated by the significantly lower number of LLM calls to make (batch size 64 requires up to 1/64 calls compared to batch size 1, the typical value used in NLP4RE research).

2.2. Automated scriptless testing of user stories (WP2)

WP2 focuses on devising scriptless testing algorithms that automatically generate tests that cover user stories as well as their acceptance criteria. The baseline for WP2 is the TESTAR tool [7], which explores software systems via agents that use an action selection mechanism. TESTAR generates test sequences of (state, action)-pairs by starting up the System Under Test (SUT) in its initial state and then continuously selecting an action to bring the SUT in another state.

In AUTOLINK, we are conducting research on how to guide action selection through the use of user stories and acceptance criteria. Initial work in this direction has explored the use of LLMs to guide the action selection mechanism [19]. We developed an improved version of TESTAR[1] where an LLM processes a so-called test goal to determine which action to take next. The test goal includes (i) a textual description of what the user would like to achieve, which is formulated in a way that resembles the 'I want' part of a user story, and (ii) a number of checks that are used to determine if the goal has been achieved. For example, a description could be *Log in with the username 'test' and the password 'star'* and a check could be *Welcome John Smith.*

We experimented with different language models (Llama3.1-8B, Llama3.2-3B, Mistral0.3-7B, Qwen2.5-7B, and GPT-4o-mini) and found that, like in the research efforts presented earlier in this paper, the GPT model performed significantly better, completing the action correctly most often (over 50 runs). In five goals, GPT's success rates were 100%, 94%, 100%, 98%, and 46%; the 46% corresponds to a long-form goal containing diverse types of widgets. Smaller models struggled more to achieve the goal; therefore, in order to use small language models, we should re-run the goal-solving test execution for the same goal multiple times, which will reduce the energy efficiency savings of these smaller models.

3. Ongoing and Future Directions

While the results in Section 2 pertain predominantly to one of the two disciplines we investigated, our ongoing work has an increasing focus on bridging requirements engineering and software testing.

From User Stories to UI Patterns. We are exploring the identification of UI patterns (also know as: UI Design patterns or HCI Design patterns) that are related to certain user stories, i.e., that could be relevant for the implementation of a user story. The working assumption is that various UI patterns (e.g., calendar picker, inplace editor, breadcrumbs, slideshow, frequently asked questions) can be tested in standardized ways; as such, if we are able to link a user story to its relevant patterns, we may be able to direct the GUI testing activities based on the typical way of testing those patterns. For example, a typical way of testing a calendar picker for a start–end date would be that of setting an end date that precedes the start date. We have assembled a catalog of over 100 patterns (collected from online resources as well as a book [20]), and we are using prompted LLMs to create a recommender system that can provide possibly relevant patterns to a developer. To assist the recommender system in the exploration of the catalog, we are experimenting with the use of a taxonomy that can be used to filter the patterns on the basis of their functional intent (e.g., informative, layout and readability, social interaction), and whether they support data insertion or data retrieval.

Matching User Stories with Extracted Behavior. One key objective of the project is to establish trace links between the expected behavior (from user stories and acceptance criteria) and the UI of the SUT. For the latter component, we are making use of the action-state model inferred from testing the system at the UI level with TESTAR [7, 21]. Given a requirement (a user story and/or acceptance criteria) and an action-state model that is inferred from multiple executions of the SUT, we aim to identify the paths that are most relevant to the requirement. We are exploring multiple automation techniques, ranging from classical information retrieval to LLMs, to perform this matching process. The major challenge lies in the abstraction gap between the artifacts: requirements indicate what the system shall achieve, while the action-state model portrays one specific way of achieving that end, without revealing what was the end.

Measuring Energy Consumption. In our earlier work, we presented initial observations on the trade-off between energy efficiency and accuracy. These experiments, however, did not explicitly quantify the energy consumption associated with computational resource usage. Thus, our ongoing

[1]LLM-powered TESTAR: https://github.com/TESTARtool/TESTAR_dev/tree/autolink

research incorporates energy consumption as an additional experimental dimension, which is an essential factor in projects that rely heavily on computational resources.

User-story Driven Testing. Our published work [19] makes use of an LLM to direct the action selection mechanism of TESTAR. However, to do so, it requires a precise goal description and corresponding checks, which act as a test oracle. However, this technique is not fully unobtrusive, since it requires humans to define the goal description and the checks, which are not generally part of the documented requirements. The next step in this direction, which will be one of the ultimate results of the project, is the direct use of information contained in the issue tracking systems (like user stories and their acceptance criteria) for the action selection mechanism. The informal and partial nature of these artifacts is obviously a challenge, which we aim to mitigate through the various techniques, including classical NLP and more recent language models, for extracting information from the requirements (WP1).

In-vivo Studies. So far, we have made use of data from industrial partners, and we benefitted from the bi-yearly project meetings. However, we could not yet test the techniques in real projects. We are planning to conduct studies in which our techniques, or similar implementations based on the same ideas, are applied and evaluated at the partner organizations. This is facilitated by their increasing adoption and investment in AI. For example, ING has invested in the creation and release of a test automation platform[2], while Mendix has included an AI system for generating low-code applications (including domain models) starting from a short description of the requirements[3].

4. Conclusion

In this project report, we have summarized the work we are conducting in the AUTOLINK project, which aims to support the automated testing of software systems at the UI level, through the use of NLP that extracts information from requirements artifacts. To reach AUTOLINK's ultimate objective, we have conducted and are performing research that makes use of NLP to (i) extract information from the lightweight requirements representations that are employed in agile development; and (ii) guide the scriptless testing at the GUI level through the definition of simple test goals in natural language.

The project started in 2023, and this coincided with the upsurge of Generative AI, thanks to the public release of ChatGPT 3.5. This event, which was quickly followed by the release of many other LLMs, rapidly affected research in software engineering [22]. This technology was adopted in AUTOLINK too and it has been a game changer. On the one hand, it simplified tasks that were deemed as very difficult (e.g., guiding TESTAR in the choice of the test actions to take with a simple textual description) and it reduced the need for training data (for example, for extracting and classifying requirements in issue tracking systems). On the other hand, GenAI poses challenges to confidentiality, as not all companies trust GenAI providers on not using their data, and it raises sustainability concerns, especially when used for tasks that could be executed through classical NLP.

We make a call to action to the research community: more research at the intersection of requirements engineering and software testing is needed, as these disciplines are interwoven and, together, contribute to software quality. First, in addition to further work on agile artifacts and GUI testing, we encourage research that uses different artifacts and testing types. Second, since GenAI techniques come with non-negligible energy requirements, we invite researchers to report simultaneously on the efficacy of the techniques and on the energy consumption. Third, we need benchmarks that consist of requirements, test cases, and corresponding software systems, as these are fundamental for validating the research.

[2]INGenious platform: https://github.com/ing-bank/INGenious
[3]Maia AI: https://www.mendix.com/platform/ai/aiad/

Acknowledgments

Thanks to all the companies that participate in the AUTOLINK project and that enable our research. This research is partially funded by the Dutch Research Council (NWO) through the Open Technology Programme 2021-II TTW, project AUTOLINK (19521).

Declaration on Generative AI

The author(s) have not employed any Generative AI tools for the writing process of this paper.

References

[1] D. Berry, R. Gacitua, P. Sawyer, S. F. Tjong, The case for dumb requirements engineering tools, in: International Working Conference on Requirements Engineering: Foundation for Software Quality, volume 7195 of *LNCS*, Springer, 2012, pp. 211–217.

[2] V. Garousi, M. Felderer, M. Kuhrmann, K. Herkiloğlu, S. Eldh, Exploring the industry's challenges in software testing: An empirical study, Journal of Software: Evolution and Process 32 (2020) e2251.

[3] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, D. Thomas, Manifesto for agile software development, https://agilemanifesto.org/, 2001.

[4] M. Cohn, User Stories Applied: for Agile Software Development, Addison Wesley, 2004.

[5] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, S. Brinkkemper, The use and effectiveness of user stories in practice, in: International Working Conference on Requirements Engineering: Foundation for Software Quality, volume 9619 of *LNCS*, Springer, 2016, pp. 205–222.

[6] E. Alégroth, R. Feldt, P. Kolström, Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing, Information and Software Technology 73 (2016) 66–80.

[7] T. E. Vos, P. Aho, F. Pastor Ricos, O. Rodriguez-Valdes, A. Mulders, TESTAR–scriptless testing through graphical user interface, Software Testing, Verification and Reliability 31 (2021) e1771.

[8] V. Tawosi, A. Al-Subaihin, R. Moussa, F. Sarro, A versatile dataset of agile open source software projects, in: International Mining Software Repositories Conference, 2022, pp. 707–711.

[9] L. Montgomery, C. Lüders, W. Maalej, An alternative issue tracking dataset of public Jira repositories, in: International Mining Software Repositories Conference, 2022, pp. 73–77.

[10] A. T. van Can, F. Dalpiaz, Requirements information in backlog items: Content analysis, in: International Working Conference on Requirements Engineering: Foundation for Software Quality, volume 14588 of *LNCS*, Springer, 2024, pp. 305–321.

[11] A. T. van Can, F. Dalpiaz, Locating requirements in backlog items: Content analysis and experiments with large language models, Information and Software Technology 179 (2025) 107644.

[12] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, S. Brinkkemper, Improving agile requirements: the Quality User Story framework and tool, Requirements Engineering 21 (2016) 383–403.

[13] M. Bragilovski, A. T. van Can, F. Dalpiaz, A. Sturm, Deriving domain models from user stories: Human vs. machines, in: International Requirements Engineering Conference, IEEE, 2024, pp. 31–42.

[14] G. Lucassen, M. Robeer, F. Dalpiaz, J. M. E. M. van der Werf, S. Brinkkemper, Extracting conceptual models from user stories with Visual Narrator, Requirements Engineering 22 (2017) 339–358.

[15] M. Bragilovski, A. T. van Can, F. Dalpiaz, A. Sturm, Leveraging machines to derive domain models from user stories, Requirements Engineering 30 (2025).

[16] C. Y. Chou, F. B. Aydemir, F. Dalpiaz, A comparative study of large and small language models for domain model extraction, in: International Working Conference on Requirements Engineering: Foundation for Software Quality, 2026.

[17] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc, Automated classification of non-functional requirements, Requirements Engineering 12 (2007) 103–120.
[18] A. T. van Can, F. B. Aydemir, F. Dalpiaz, One size does not fit all: On the role of batch size in classifying requirements with LLMs, in: International Workshop on Artificial Intelligence for Requirements Engineering, IEEE, 2025.
[19] C. Van Hooren, F. P. Ricós, S. Bromuri, T. E. Vos, B. Marín, LLM-Empowered Scriptless Functional Testing, in: Intl. Conference on Software Quality, Reliability and Security, IEEE, 2025, pp. 1–12.
[20] J. Tidwell, Designing interfaces: Patterns for effective interaction design, O'Reilly Media, 2010.
[21] A. Mulders, O. R. Valdes, F. P. Ricós, P. Aho, B. Marín, T. E. Vos, State model inference through the gui using run-time test generation, in: International Conference on Research Challenges in Information Science, Springer, 2022, pp. 546–563.
[22] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, H. Wang, Large language models for software engineering: A systematic literature review, ACM Transactions on Software Engineering and Methodology 33 (2024) 1–79.