Scientific Iniciation Proposal

DeeperMatcher: using LLM for Crowd Based Requirements Engineering

Arthur Pilone

Prof. Paulo Meirelles (Advisor)
Prof. Fabio Kon (Co-Advisor)
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
arthurpilone@usp.br,{paulormm,kon}@ime.usp.br

Prof. Walid Maalej (International Supervisor)
University of Hamburg
walid.maalej@uni-hamburg.de

Abstract

One of the most complex challenges in ensuring software quality is assuring the convergence of the developers' and users' views. Requirements Engineering studies how this can be achieved by investigating how software requirements can be collected and maintained. Nevertheless, it is still unclear how development teams can take advantage of the large amounts of user data found on various social media, app store reviews, and support channels. In this study, we aim to develop and empirically investigate a Machine Learning-powered tool called DeeperMatcher, tailored for agile software development teams to use crowd-based requirements engineering (CrowdRE) to aid the management of their issues and tasks. The research unfolds across three objectives: (I) developing a reliable, maintainable, and organized ML-enabled system; (II) leveraging advancements in natural language processing to provide an approach for CrowdRE; and (III) applying empirical research methods for system validation. Methodologically, we will incorporate a single-case mechanism experiment with a real-world dataset from a specific project developed by our research group and observational case studies from different projects. Our execution plan comprises two phases, the first emphasizing the tool development and validation and the second dedicated to extensive testing and in-depth analysis. Amidst our expected outcomes, we list a well-structured AI-powered system, noteworthy contributions to software and requirements engineering, and valuable insights into the evolving landscape of machine learning in software development.

1. Justification

With the growing complexity of the computational systems developed since the second half of the 20th century, it became clear that there would need to be established values, methods, and practices to ensure (i) the usability, stability, and correct function of software systems; (ii) that the vision development teams and end users have for the product being developed converge; (iii) the extensibility and maintainability of the software system, while still preserving items (i) and (ii). However, before the end of the 1960s, and shortly after the creation of the first large-scale computational systems, the field of **Software Engineering** began to grow (Valente, 2020). Its main intention would be to secure these three points by employing processes and guidelines empirically consolidated by years of practice over other engineering fields.

Among the different software engineering areas, **Requirements Engineering** is the one responsible for sustaining the abovementioned principles. The primary focus is the identification and analysis of **requirements**. As defined in the IEEE (1990) standard, a requirement is "*a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents*". On the other hand, the fundamentally subjective nature of what makes a software product adequate to its users poses challenges to approaches based on the principles of what we understand as "traditional software engineering".

Born at the turn of the century, agile software development methods emerged as a countermovement inside software engineering, advocating for a more human and personal approach to formulating the values and practices that guide the development of software systems. Among the practices that summarize their principles are (i) prioritizing constant and recurring feedback from the users and stakeholders, (ii) separating the development period into short cycles, each with a set of tasks to be done before its end, and (iii) the concept of writing requirements in the form of *user*

stories, which pushed developers to think about the users a given task addresses by having each task registered as a story written by a hypothetical user describing how and why they would use the relevant feature (Beck and Andres, 2004; Julian et al., 2019).

Although undeniably valuable for the development team and still considered a proper way of recording a requirement, writing user stories may not be feasible for every software product specification over time. Moreover, the prospect of having a customer participate and contribute to the team by writing them is often far from what is possible in many software products. Hence, it is not rare to see teams using other formats to keep track of the project requirements and tasks to be done. This convenience over the process of recording and maintaining the team tasks comes with the cost that a given requirement may not be helpful for the software users and the risk that the tasks the development team writes and the actual needs of the product users begin to drift apart.

This scenario may be even more challenging for smartphone apps with larger user bases. A requirement related to a specific group can probably involve other users, but it may never get to the development team due to the absence of direct communication between developers and users. It is not hard to see how the problem is exacerbated when it comes to bug and incident reporting. The vast range of operating systems, device models, execution environments, and user contexts make the feat of accurately predicting how a given software product behaves in every possible condition nearly unattainable.

Amidst thousands of user reviews and comments on app stores and social media, prioritizing which comments should be studied by the development team as starting points for new requirements is challenging and demands an unreasonable amount of human analysis. The prospect of tapping into the considerable amount of data embedded in these reviews and posts on social media has been named **Crowd-Based Requirements Engineering** (CrowdRE) (Groen et al., 2017; Snijders et al., 2014).

This approach has been studied before, especially using Machine Learning models and Natural Language Processing (NLP) (dos Santos et al., 2019; Mekala et al., 2021; Stanik et al., 2019), with varying scales of databases and levels of accuracy.

After selecting which user requirements are to be implemented, the team is still supposed to register them unambiguously, correctly conveying the users' needs and perception of the requirements. Managing such an extensive collection of requirements also comes with the challenge of avoiding the replication of issues on the team backlog, as well as dealing with tasks that may intersect, which raises the complexity of maintaining the issue tracker (Fernández et al., 2016; Uludag et al., 2018).

Haering et al. (2021) proposed a tool called *Deep Matcher* that automatically matches problem reports in app reviews to bug reports in issue trackers using machine learning methods. The initial version was a proof of concept that covered four databases with issues and reviews written in English, pointing towards a promising application of NPL in requirements engineering. Table 1 represents an example of how the matches found by *DeepMatcher* may be represented.

Problem Report	**Suggested Bug Report Summary**
APP: VLC **Date:** 2020-05-17 **Report:** So many bugs... Plays in background, but no controls in notifications. When you tap the app to bring up the controls, the video is a still screen. Navigating is a pain. Resuming forgets my place constantly. Basically unusable	**Date:** 2019-04-13 **Report:** android navigation bar, shown after a click, shifts and resizes full-screen video
	Date: 2018-09-27 **Report:** Play/pause button icon is not shifting while pausing the audio on notification area
	Date: 2013-09-16 **Report:** [Android] On video playing the navigation bar is not hidden on some tablets

Table 1: Example matches associating issues with bug reports (Haering et al., 2021).

Prof. Walid Maalej, from the University of Hamburg, a co-supervisor of this research project, is a co-author of the paper mentioned above. He is an award-winning researcher and one of the most prominent figures of the new generation in software and requirements engineering. He has given us access to the *Deep Matcher* paper replication package, and with his collaboration, we will further enhance the

system his group proposed.

In addition to implementing *Deep Matcher* as a proper free/open-source tool, we will extend it by using the latest NLP advances, and will call it **DeeperMatcher**. Based on the original approach (Haering et al., 2021), we will use machine learning to aid agile software development teams in recording and maintaining issues and tasks based on large amounts of user feedback. Additionally, we will analyze the effectiveness of using these technologies on the requirement engineering processes.

2. Objectives

Our main goal is to empirically explore the feasibility of designing and implementing the Deeper Matcher tool for agile software development teams to better treat large amounts of user feedback with a Machine Learning model as the core of its architecture. The increased use of ML elements in large systems brings new challenges for software engineering, mainly centered around assuring a reasonable standard for the system effectiveness, accuracy, and predictability while based on a fundamentally non-deterministic and uncertain component (Serban and Visser, 2022).

Besides studying the problem from a software architecture standpoint, we will evaluate and validate DeeperMatcher, guaranteeing product maintainability and the interchangeability of the models used in the tool. As the field of artificial intelligence focused on Natural Language Processing (NLP) has received and is still receiving significant attention in recent years, one of our objectives is to enable different Large Language Models (LLMs) to be integrated into DeeperMatcher for classifying users feedback and embedding bug reports. We plan to analyze DeeperMatcher's performance over the datasets used in the original paper (Haering et al., 2021). We also intend to investigate the flexibility and generalization potential of the most advanced ML models released after the original work of Haering et al. (2021).

3. Methodology

We will develop the DeeperMatcher in the context of the InterSCity project[1], which studies solutions at the intersection of software engineering, data science, and distributed systems in applications such as public health care, urban mobility, and public policy-making for future smart cities. In this context, our research group is leading the "Bike SP" pilot project[2]. "Bike SP" is a program enacted by the São Paulo city council in 2016 that would financially reward citizens who registered daily bicycle commutes[3]. However, the approved law is unclear regarding how cyclists should record their trips and how much the city would pay them for each. Thus, to help solve these gaps, our group was asked by the Municial Secretariat for Mobility and transportation to develop a mobile app capable of recording users' trips and validating whether they made them by bicycle. The application would be used in a pilot program with hundreds of selected volunteers to aid the Bike SP program's planning and other public policies to incentivize cycling in São Paulo (Lima, 2023).

As the mobile app for the Bike SP pilot program is entering its testing phase, the development team is starting to receive substantial feedback from its beta testers. Therefore, besides being a possible source of data for us to use during DeeperMatcher implementation, the tool could also assist the Bike SP app developers in maintaining their issues while they test the system.

The first phase of this project consists of rewriting the DeepMatcher initial code base and building DeeperMatcher (Section 3.1). The second phase brings a scientific approach to analyze and validate DeeperMatcher (Section 3.2). We will conduct a *single-case mechanism experiment* (Wieringa, 2014b), testing and validating the DeeperMatcher using the dataset of users' feedback and development tracking issues from the Bike SP pilot app. Afterward, we will test DeeperMatcher further and quan-

[1] https://interscity.org/
[2] https://interscity.org/bikesp/piloto/
[3] https://legislacao.prefeitura.sp.gov.br/leis/lei-16547-de-21-de-setembro-de-2016

tify its accuracy and usefulness by conducting *observational case studies* (Wieringa, 2014a) with multiple datasets from other apps.

3.1. Software Development Method

We will implement the DeeperMatcher following agile software development values and practices. We will split the time allocated to the development into 2-week-long sprints, each with a different set of tasks and improvements to be done to the system' current state.

We will employ a subset of the dataset originating from users' feedback on the beta version of the Bike SP pilot app to elaborate automated tests built to guide and validate the project implementation. Additionally, we will use part of the data for applying *test-driven development* (TDD) over the system implementation.

All code from this research project will be distributed as free/open-source software. This way, anything created in this research project is left as a contribution to the research in software requirements engineering and to the software development community in general.

3.2. Research Method

In the first research phase to validate DeeperMatcher, we will carry out a single-case mechanism experiment based on users' feedback and issues from the Bike SP pilot project, which will be processed and fed into the system to validate its proposed architecture and implementation. Focusing on a single dataset, we will investigate how the application behaves when dealing with data from a real-world context. Besides validating the implementation, this approach will provide system performance estimates, instructing DeeperMatcher's development and evolution.

In the second phase, following the conclusion of the DeeperMatcher implementation, we will perform a couple of observational case studies to examine and analyze

the matches[4] it generates when applied to additional datasets. Among the cases to be studied, we include the ones already employed in the original DeepMatcher paper to measure if and how the enhancement affects the metrics initially obtained in the previous study (Haering et al., 2021).

We will interpret the matches found by the DeeperMatcher from the data collected in our observational case studies. Besides measuring the frequency in which our tool accurately finds an existing issue for a given bug report, we will also examine how it could be used when the system cannot find a corresponding issue for a given bug report.

4. Execution Plan

This project involves developing the DeeperMatcher and applying empirical research methods for system validation: a single-case mechanism experiment and observational case studies.

To start the development of DeeperMatcher, from the replication package code of the original DeepMatcher paper (provided by Haering et al. (2021)), we will carry out a major refactoring. We will separate the parts used for analyzing the results and treating its data from the ones responsible for classifying the textual feedback and embedding and matching issues and reports. This way, we envision the delimitation of a "core logic" for the DeeperMatcher tool, independent from the data used and treated by the system.

Following this, we will decouple the issue and feedback collectors, making the system extensible for new data sources. We will use the same strategy for the parts that encapsulate the machine-learning models for classifying user feedback and embedding issues. It is essential to decouple these components to make the system adaptable to the change in the model, favoring DeeperMatcher's maintainability.

[4]Matches from user bug reports to issues in the issue tracker.

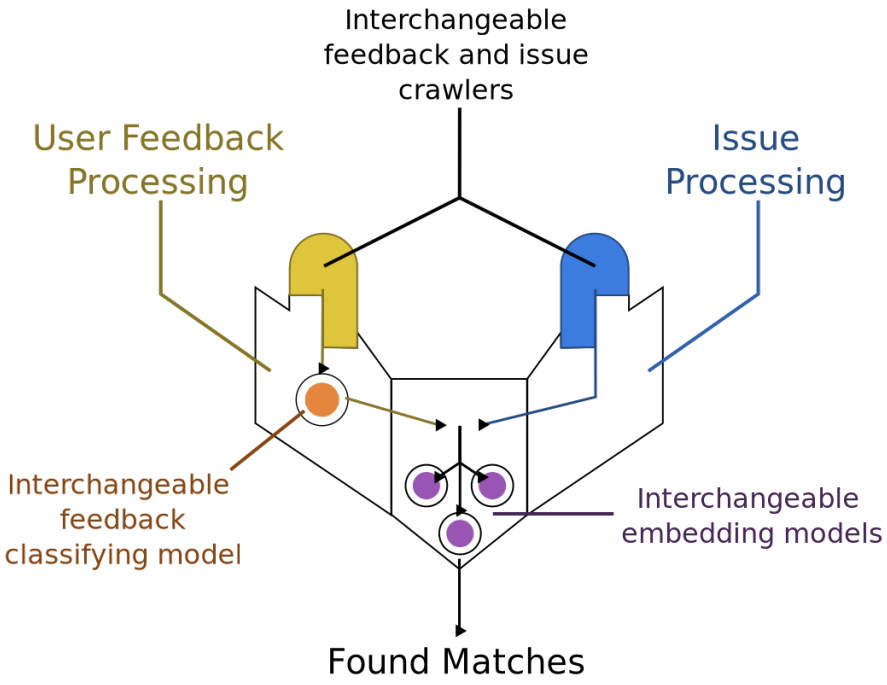

Figure 1: Fundamental elements of DeeperMatcher's architecture.

Figure 1 illustrates the main characteristics of DeeperMatcher's architecture as described in the previous paragraphs. In yellow and blue, we depict the issue and feedback collectors. The central part of our figure contains the module housing the tool's "core logic", which receives the data treated by the issue and feedback processors. We also represent the machine learning models used in the architecture with colored circles to indicate that they are components that DeeperMatcher's maintainers may replace in the future.

In the meantime, we will collect, compile, and label the feedback from the Bike SP pilot app testers in preparation for the single-case mechanism experiment. We will conduct this experiment along with the system implementation, identifying possible

faults and points of improvement as they arise; thus, as the initial development phase ends, we may validate the integrations between the DeeperMatcher components (as illustrated in Figure 1).

As the system is expanded to house new issues and user review collectors, we plan to implement a non-relation (NoSQL) database system responsible for adequately providing persistence to the multi-form data collected from different sources.

In the second phase of this project, we will conduct observational case studies in which we will test DeeperMatcher using different datasets. Besides the ones used in the original DeepMatcher paper, we plan to select new data sources from software products with large amounts of publicly available issues and user feedback (including in Portuguese). Finally, we will analyze and interpret the results from the case studies, verifying whether the resulting mechanism could be helpful for software development teams and how so.

Activity	B1	B2	B3	B4	B5	B6
System Re-Implementation	•	•				
Addition of new data collectors		•	•			
Single-Case Mechanism Experiment		•	•			
Conduction of the Observational Case Studies			•	•	•	
System Persistence Extension				•	•	
Research internship @USI				•	•	•
Analysis of the Case Studies					•	•
Paper / Article Writing						•

Table 2: Proposed schedule (in bimesters)

In Table 2, we detailed the proposed execution schedule as divided into bimesters, starting on February 1st, 2024. We plan on employing our first two bimesters to evolve the code from DeepMatcher's proof of concept into our vision for DeeperMatcher.

While the first of these two bimesters will be dedicated exclusively to the system's development, starting in the second bimester, we will perform our single-case mechanism experiment and conclude it before the end of the third bimester (July

2024). At the same time, we will extend DeeperMatcher to work with user feedback and issues written in Portuguese and collected from new issue crawlers.

Beginning in June 2024 (By bimester 3), we will use six months to conduct our observational case studies, and, beginning in September 2024, we will dedicate 3 months to adapting DeeperMatcher's issue and feedback persistence modules, namely implementing a non-relational database for storing the collected data.

It is worth noting that, as we are writing this revision to the project, a 4-month research internship at the USI (Università della Svizzera italiana) is being considered. Starting in September, the visit aims to bring into the project the expertise of the professors at one of the best Software Engineering research groups in Europe. Finally, we allocate our last bimester (December 2024 - January 2025) to analyzing the results of the observational studies conducted in the six previous months.

5. Expected Results

According to our objectives, we expect several results from our research project. With DeeperMatcher, we will provide a system adequately organized to house an artificial intelligence component. The source code will be available as free/open-source software from its first code commit, accessible to anyone who wants to study, modify, or evolve it. As a result, we expect to leave behind a pathfinder indicating how developers may extract the benefits of Crowd-Based Requirements Engineering to help their issue management.

Additionally, by creating an architecture that enables the interchangeability of the ML models used and measuring a reasonable level of the DeeperMatcher's accuracy when we fit it with newer models, we will have attested the feasibility and practicality of building and utilizing such a system.

Our single-case mechanism experiment aims to methodically characterize how DeeperMatcher generates a list of matches when provided different inputs from a

real-world dataset. We will understand what to expect from each system component as we change the characteristics of the input data. Most importantly, we will verify that DeeperMatcher works for issues and textual feedback written in Portuguese.

Finally, through our observational case studies, we expect to reproduce the results achieved by Haering et al. (2021) with quantitative metrics comparable to or slightly better than those observed in the original publication. Besides measuring these differences in the *Mean Average Precision* and *Hit Ratio* of DeeperMatcher's outputs for DeepMatcher's replication data, we expect similar results for other datasets, such as the one from the Bike SP pilot app.

References

Beck, K. and Andres, C. (2004). *Extreme programming explained: embrace change*. Addison-Wesley Professional.

dos Santos, R. I., Groen, E. C., and Villela, K. (2019). An overview of user feedback classification approaches. In *REFSQ Workshops*.

Fernández, D. M., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., Conte, T., Christiansson, M., Greer, D., Lassenius, C., Männistö, T., Nayabi, M., Oivo, M., Penzenstadler, B., Pfahl, D., Prikladnicki, R., Ruhe, G., Schekelmann, A., Sen, S., Spínola, R. O., Tuzcu, A., de la Vara, J. L., and Wieringa, R. J. (2016). Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice. *CoRR*, abs/1611.10288.

Groen, E. C., Seyff, N., Ali, R., Dalpiaz, F., Doerr, J., Guzman, E., Hosseini, M., Marco, J., Oriol, M., Perini, A., and Stade, M. (2017). The crowd in requirements engineering: The landscape and challenges. *IEEE Software*, 34(2):44–52.

Haering, M., Stanik, C., and Maalej, W. (2021). Automatically matching bug reports with related app reviews. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 970–981.

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, page 62.

Julian, B., Noble, J., and Anslow, C. (2019). Agile practices in practice: Towards a theory of agile adoption and process evolution. In *Agile Processes in Software Engineering and Extreme Programming*, pages 3–18. Springer International Publishing.

Lima, A. (2023). Cycling promotion using financial incentives: A pilot design to inform public policy in são paulo, brazil. https://linux.ime.usp.br/~analima/mac0499/assets/pdfs/report.pdf.

Mekala, R. R., Irfan, A., Groen, E. C., Porter, A., and Lindvall, M. (2021). Classifying user requirements from online feedback in small dataset environments using deep learning. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 139–149.

Serban, A. and Visser, J. (2022). Adapting software architectures to machine learning challenges. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 152–163.

Snijders, R., Dalpiaz, F., Hosseini, M., Shahri, A., and Ali, R. (2014). Crowd-centric requirements engineering. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 614–615.

Stanik, C., Haering, M., and Maalej, W. (2019). Classifying multilingual user feedback using traditional machine learning and deep learning. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 220–226.

Uludag, O., Kleehaus, M., Caprano, C., and Matthes, F. (2018). Identifying and structuring challenges in large-scale agile development based on a structured literature review. In *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 191–197.

Valente, M. T. (2020). *Engenharia de Software Moderna*.

Wieringa, R. J. (2014a). *Observational Case Studies*, pages 225–245. Springer Berlin Heidelberg, Berlin, Heidelberg.

Wieringa, R. J. (2014b). *Single-Case Mechanism Experiments*, pages 247–267. Springer Berlin Heidelberg, Berlin, Heidelberg.