DevMetrics: An Experience Report on Effort Estimation in Software Maintenance Using a Jira-Integrated Framework

Marcos Andrade*
marcospandrade2@gmail.com
Federal University of Technology – Paraná (UTFPR)
Dois Vizinhos, Paraná, Brazil

Marisangela Brittes*
mbrittes@utfpr.edu.br
Federal University of Technology – Paraná (UTFPR)
Dois Vizinhos, Paraná, Brazil

Alinne Souza*
alinnesouza@utfpr.edu.br
Federal University of Technology – Paraná (UTFPR)
Dois Vizinhos, Paraná, Brazil

Jessica Pegorini*
jessicapegorini@utfpr.edu.br
Federal University of Technology – Paraná (UTFPR)
Dois Vizinhos, Paraná, Brazil

ABSTRACT

This paper presents the development of DevMetrics, a generic system designed to support effort estimation in software maintenance activities. The tool combines textual similarity (Jaccard index) with statistical regression (linear and exponential), normalized using the Fibonacci sequence, and integrates seamlessly with Jira. A case study was conducted in a medium-sized software house using MMRE and PRED(0.25) metrics, along with Likert-based surveys to assess team perception. Results show that the tool enhances estimation accuracy and planning predictability, especially when teams adopt standardized task descriptions and ensure historical data quality.

KEYWORDS

Software Quality Assurance, Software Maintenance and Evolution, Software Measurement, Software Process Improvement, Agile Software Development Methods, Software Processes, Methods, and Tools

1 Introduction

Companies that adopt agile methodologies, especially those that provide software consulting or operate as custom software builders, heavily depend on effort estimates that closely reflect reality in order to maintain a profitable operation. This allows them to deliver greater predictability to their clients regarding deliverables, particularly in the context of software maintenance, which is estimated to represent around 90% of the total cost of a software project [8].

In the context of small and medium-sized enterprises, this metric often goes unnoticed due to the lack of structured processes. However, such oversight can introduce additional costs to the business by leading to flawed software delivery planning. Frequently, these companies rely heavily on the empirical experience of their developers to estimate tasks, some of which may have been previously executed by the team in another context. Additionally, when dealing with highly specific use cases, it's common to encounter overly optimistic estimates.

The issue with relying on empirical estimates is that they are based on the subjective judgment of those estimating the tasks, leading to planning inconsistencies and parameters that are not always trustworthy. Leveraging historical data makes it possible to generate estimates that are more aligned with the team's actual track record in executing planned maintenance tasks and feature implementations, thus transforming an empirical process into a standardized one. However, manually mining historical data can introduce unnecessary overhead into team planning, which may have the opposite effect.

Software maintenance further increases complexity, as it often involves estimating refactoring effort [8], which can be hindered by lack of system documentation or inadequate test coverage, both of which make accurate maintenance estimation more difficult.

Given this scenario of uncertainty and unpredictability, it is often challenging to present maintenance cost estimates to clients [5]. With data that more accurately reflects reality and better expectation alignment, it becomes possible to demonstrate medium- and long-term benefits that justify the maintenance effort within a project's scope.

In this context, this work aims to provide a reference for development teams to estimate maintenance tasks or new feature development with greater precision and predictability. This is achieved through a strategy that scores historical project tasks based on textual similarity. The objectives of this study include: (i) proposing a generic tool inspired by the concept introduced by [14]; (ii) automated integration with Jira; (iii) combined use of textual similarity and statistical regression to generate estimates for new projects; and (iv) real-world effectiveness evaluation using historical data.

This tool is designed to assist project managers and/or Scrum Masters by providing data-driven support for effort estimation based on the team's historical records. While following the idea proposed by [16], this tool is developed as a more generic and customizable solution that integrates with one of the most widely used project management tools, Jira.

Estimates in the tool are generated automatically by evaluating historical tasks similar to the ones selected and applying statistical regression to estimate future tasks. To assess the tool's effectiveness, a case study was conducted involving both qualitative and quantitative evaluations within a small-sized software house.

2 Related works

In the context of software development, changes are inevitable. Therefore, it is important to have mechanisms to evaluate, control, and execute modifications [12]. However, in an environment with continuous change throughout the development cycle, it is necessary to use methodologies and tools to ensure that these activities are properly organized within the project scope.

These changes must be planned so that the process occurs in the most accurate and efficient way, using appropriate tools and methods [6]. Within this planning process, effort estimation for executing process activities is essential, as these parameters are key for agile methodologies to reach their full potential.

Estimates generated solely on empirical data are subjective and have low accuracy, and the commercial solutions available for integration with project management tools generally lack structured methodological support to assist teams during development, leaving the responsibility entirely to the teams' judgment.

When considering the implications of effort estimation in software, it becomes clear that this is a complex activity. When performed inaccurately, it can result in reduced delivery quality and even cause delays or prevent software delivery altogether [15].

According to [3], there are three main groups of techniques created to assist with the effort estimation process: (i) algorithmic methods (Function Point Analysis or COCOMO); (ii) non-algorithmic methods (Planning Poker, Ideal Days); (iii) machine learning-based methods (neural networks, feed-forward networks, LLMs). This last group is a more recent approach, made viable by advances in machine learning over the past decade.

This work proposes the construction of a solution using algorithmic techniques aimed at greater interpretability of quantitative and historical data, relying less on empirical judgment. It also seeks simplicity, so the system can be applied by companies with less data maturity that do not maintain well-organized historical datasets, while providing more inputs to the team during the planning of maintenance activities.

Among the works reviewed, the study presented by [10] was considered. It uses the AD-Reputation metric, which incorporates developer reputation scores based on previous activities through Bayesian inference, combined with the empirical estimates provided by the team. However, this model does not directly connect reputation with task attributes such as criticality, complexity, or module.

Although it was tested in a real-world case study, the scope of the application was limited in terms of team diversity and tool usage. Therefore, a more task-attribute-centric approach was chosen, integrated into the development environment and directly connected to the team's workflow through Jira.

The system proposed in this work is based on the idea introduced by [16], who developed a tool specifically tailored to the needs of the company used in the case study. The original product connected directly to the company's internal database to retrieve historical activities, without offering any external integration. As a result, all activities the team wanted to estimate had to be entered manually. This approach limited the tool's reusability in other scenarios. Additionally, it did not accommodate contextual variations between teams and projects, which led to the decision to completely rewrite the tool in version 2.0 (DevMetrics), enhancing the original idea with new features and different estimation techniques, as shown in Table 1.

Given the limitations previously identified, this work proposes DevMetrics, developed as a generic, reusable tool that integrates directly with Jira. It was built to reduce the manual intervention identified as a problem in the initial proposal [16]. By integrating with the user's project, the tool allows the creation of sprints and the generation of estimate suggestions based on algorithmic methods such as textual similarity and linear regression, in an automated way, with data persistence and minimal friction in the adoption process by agile teams.

The use of textual similarity techniques combined with statistical regression is justified by the need for traceability and explainability in the estimate generation process. Unlike more complex models, where decision logic is often opaque, the chosen approach makes it possible to identify which elements were or were not used in producing a given result.

Moreover, by establishing a documentation standard that pairs tasks with similar descriptions, it becomes easier to understand the activity context, especially for teams with lower analytical maturity.

3 Proposed tool

This work proposes DevMetrics, an evolution of EstimAi developed by [16], with a focus on making the platform more generic, reusable, and accessible. It is a redesigned web system that offers a flexible interface capable of serving different companies and facilitating onboarding for new users through low-effort integrations.

The tool aims to support agile teams during the effort estimation stage, especially for software maintenance tasks, by using historical data from previously executed tasks to establish a baseline for new estimates. This reduces the manual effort required from project managers and increases planning predictability.

Users can log into the web system using their Jira account, and the system authenticates with Atlassian using the *OAuth2* strategy. This is an authorization protocol that allows applications to access data from an external service on behalf of the user, securely and without exposing credentials.

This approach eliminates manual registration and persists data related to the user's selected project. This design was chosen to simplify the integration of new users into the platform.

3.1 Components and Architecture

DevMetrics is composed of two main services: a *frontend* built with Next.js, a React.js framework, and a central *backend* built with Nest.js, a Node.js framework that handles data persistence and estimate generation. They communicate via internal REST APIs, and authentication and access control are handled using *OAuth2* tokens obtained through secure integration with the Atlassian API [1]. OAuth2 authentication ensures secure access to jira data without storing any business logic, only metadata such as jira identifier, title and story points.

The integration with Jira begins with the frontend sending an authentication request for an Atlassian account. After communication with the service, a code and an access identifier property called *state* in UUID format are returned. The frontend then sends this data to the server, which holds the credentials needed to convert the code into a Jira access token and stores it internally.

The server then accesses the necessary information to perform the initial project setup in DevMetrics and generates an internal *JWT (JSON Web Token)* to be returned to the frontend. Only the server is authorized to make calls to Atlassian, following web security best practices.

Table 1: Comparison between the original framework [16] and DevMetrics

Aspect	Viesseli (2021)	DevMetrics (this work)
Scope	Specific solution for a single company	Generic and reusable framework for multiple contexts
Integration with tools	No external integration	Automated integration with Jira via OAuth2 API
Activity registration	Manual, duplicated in the system and in Jira	Automatic synchronization of activities directly from Jira
Data storage	Direct access to the company's database	Own database with secure and decoupled synchronization
Adherence to agile process	Limited to the specific workflow of the company	Aligned with sprint planning using story point estimates
Estimation techniques	Textual similarity (Sørensen-Dice) + simple linear regression	Textual similarity (Jaccard) + linear and exponential regression + Fibonacci normalization
Interface and usability	Basic interface, not adapted for new teams	User-friendly web interface with guided setup and customizable fields
Target audience	Specific internal team	Agile teams in different organizations using Jira
Reuse of history	Limited to local database	Reuse of historical data synchronized from Jira

The chosen database for persisting project data is *Postgres*, due to its robustness in supporting relational models and its maturity in web applications. The database structure includes *integration servers* as a core entity, representing a Jira server. This allows multiple projects from the same company to be synchronized. This modeling enables the final product to be a robust, adaptable system suitable for enterprise environments.

Another important aspect is the strategy for storing estimates in the *issues* entity, which is linked to a specific *sprint*. This allows consistency across estimates for tickets selected in different sprints, as shown at the Figure 1

Another entity required during application development was *custom fields*, since both the sprint field and the story point field are considered customizable within Jira. As the system allows administrators to personalize the fields used in tasks and the format of received estimates, a project *setup* step was introduced after synchronization so the user can specify which fields DevMetrics should use to identify sprints and estimate points.

The project synchronization flow occurs right after the user logs in. Through *OAuth2* authentication, the user selects which server to use to start the synchronization process. Once authentication with Atlassian is complete, the server receives the notification and begins synchronization.

If the server has more than one project, the system waits for the user to manually select the project. If there is only one project, the system returns success and unlocks the main flow, starting a decoupled, asynchronous process that fetches all project tickets, splits them into batches, and processes them in the background.

This configuration step ensures DevMetrics works properly even in customizable environments like Jira, maintaining consistency in the estimation process. After establishing the architecture and organizing the activities by project, the next step involves creating *sprints* and, after selecting tickets, applying textual similarity and linear regression techniques to suggest story points based on the team's historical data.

3.2 Applied Techniques

After the ticket synchronization and custom field setup, the user must register the corresponding team to associate it with a *sprint* and start generating estimates. In the future, the system may evolve to also manage team capacity for sprint planning.

From the DevMetrics homepage, users can register and synchronize all necessary information. To generate estimates, the user must create a *sprint*, select the project and team, fill in the sprint's start and end dates, and define the planned goals. This process provides an overview of the iteration's planning.

Then, the user selects which tickets to include in the sprint plan. At this stage, they can filter by issue code, as shown in Figure 2. The estimation process consists of two main steps: creating a set of reference activities and calculating estimates for the sprint tickets.

In the first step, all completed activities with team-assigned scores from previous sprints are selected from the issues table. This initial filtering is required to ensure that only historical data is used. This set becomes the baseline for estimating the sprint's tickets.

At the end of this stage, asynchronous events are triggered to individually generate estimates for each selected ticket. This decoupling improves user experience and system responsiveness.

In the second step, for each triggered event, the system separates the title and description and applies the Jaccard similarity index [13] to each. The *natural* library is used to tokenize the strings and calculate the similarity as the intersection of shared words divided by the union of all unique words in either text.

This method was chosen because the initial study using Sørensen-Dice coefficients resulted in many false positives when measuring textual similarity between task descriptions [16]. In the original

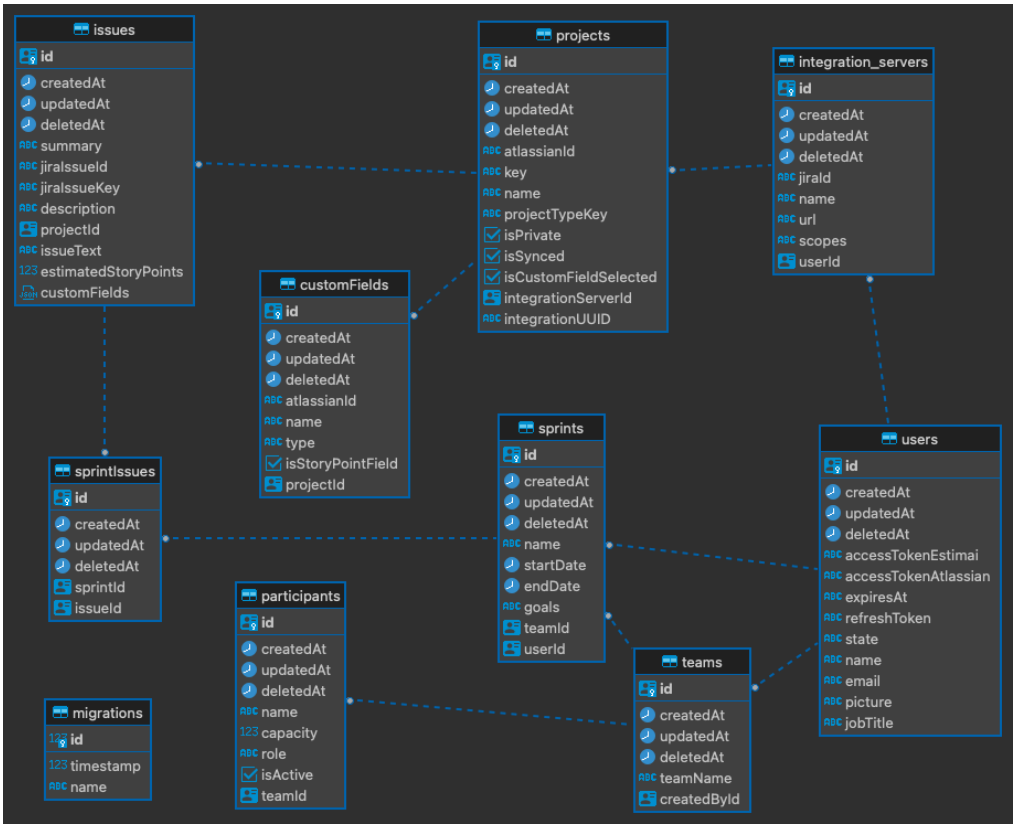

Figure 1: Database structure of the DevMetrics system.

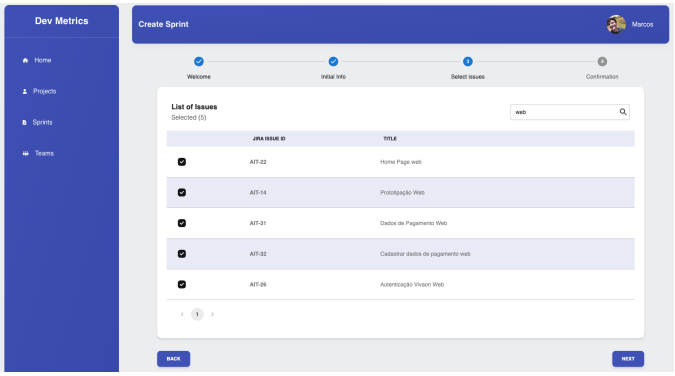

Figure 2: List of tickets during sprint creation in DevMetrics.

framework, titles and descriptions were concatenated into a single string, which may have skewed the similarity calculations.

Only activities with similarity above the minimum acceptable threshold continue to the estimate generation step. After testing several percentages, 30% similarity was chosen as the most effective cutoff point.

Selected activities serve as input for statistical regression algorithms, with a final normalization step using the Fibonacci sequence, a common practice in agile teams [7]. Both linear and exponential regressions are computed, their results averaged, and then mapped to the closest Fibonacci number to generate the final story point estimate.

The similarity score between the sprint activity and its historical counterpart is used as the independent variable (X-axis), and the story point from the historical activity is used as the dependent variable (Y-axis), as shown in Figure 3.

The combination of linear and exponential regression is intended to capture different patterns between similarity and estimated effort. While linear regression works well with stable proportional relationships, exponential regression handles abrupt variations, especially due to the sensitivity of the textual similarity algorithm.

By combining both statistical models, it is possible to obtain more accurate estimates that compensate for the individual limitations of each model when dealing with different behaviors in historical data. The *regression* and *simple-statistics* libraries were used in the final system.

After calculating similarity for both title and description, the system computes their arithmetic mean, normalizes the result to the nearest Fibonacci number, and stores the value in the database under the generated estimate field. This value is displayed to the user in DevMetrics.

Once the estimate generation process is complete, the user can view the returned values for each selected ticket along with the

Figure 3: Relationship between linear and exponential regressions used for story point prediction. (Author's Work)

sprint's main information. This way, DevMetrics offers contextualized estimates based on real historical data, providing support for sprint planning. The following section presents the results of a case study applying DevMetrics in a small software development team.

4 Validation and results

This section presents the results obtained after applying the framework in a small-sized IT company. It includes both the validation process and the findings from the case study.

The validation process followed the Goal-Question-Metric (GQM) approach, ensuring a structured evaluation through predefined indicators. Quantitative analysis included MMRE and PRED(0.25) metrics, while user perception was captured through a Likert-scale questionnaire aligned with the Technology Acceptance Model (TAM), allowing for a comprehensive view of both system performance and adoption challenges.

4.1 Case Study

At the end of the DevMetrics tool development, a case study was conducted in a real company with an exploratory nature, aiming to evaluate the effectiveness of DevMetrics in generating estimates based on historical activities and to understand user perceptions regarding the tool's impact on sprint planning. The investigation followed the Goal-Question-Metric (GQM) model [2], addressing both quantitative and qualitative indicators.

The guiding questions were: (QP1) Are the estimates provided by DevMetrics close to the actual time spent on tasks? and (QP2) What is the team's perception of using the tool during sprint planning?

To answer these questions, consolidated quantitative metrics from the literature were used, such as the Mean Magnitude of Relative Error (MMRE) and PRED(0.25), which measures the proportion of activities whose estimation error was below 25%, both derived from the MRE metric.

The Mean User Perception (MPU) was captured using a questionnaire applied at the end of the case study. The questions were presented as affirmative statements, with response options based on the Likert scale [9], ranging from 1 (strongly disagree) to 6 (strongly agree). An MPU value greater than 3 was interpreted as a positive perception of the tool.

In this work, the Technology Acceptance Model (TAM), proposed by [4], was used as a complementary theoretical lens to interpret the questionnaire responses organized under the MPU metric. This model provides a consolidated conceptual framework to understand how users perceive usefulness, ease of use, and intention to adopt new technologies.

To align the qualitative analysis with a consolidated theoretical basis, the questionnaire items were organized according to the guidelines proposed by the TAM model [4]. Table 2 presents the mapping between each question and its respective TAM dimension, enabling a structured interpretation of user perceptions regarding DevMetrics.

Table 2: Mapping of questionnaire items to TAM dimensions

Question	TAM Dimension
The estimates generated by DevMetrics were useful for sprint planning.	Perceived Usefulness (PU)
The estimates were close to the actual time required to complete the tasks.	Perceived Usefulness (PU)
Using the tool reduced uncertainty when estimating new tasks.	Perceived Usefulness (PU)
Integration with Jira facilitated the use of DevMetrics.	Perceived Ease of Use (PEOU)
Using the tool required a reasonable amount of time within the team's routine.	Perceived Ease of Use (PEOU)
Authentication and project selection were simple.	Perceived Ease of Use (PEOU)
DevMetrics fit well into the team's workflow.	Attitude Toward Use (ATU)
Using the tool helped reduce rework or revisions in the estimates.	Perceived Usefulness (PU)
I would recommend using DevMetrics in other sprints or teams.	Behavioral Intention to Use (BIU)

The case study was applied in a small-sized software development company, which focuses on software maintenance and the construction of digital products. For convenience, the selected project was one for which the company had already delivered a first version and was receiving ongoing requests for updates and maintenance of existing functionalities.

In this company, a team was selected consisting of three developers, one designer, and one product owner. The participants were intentionally chosen as they were already part of the target project team. The team works with two-week sprints, conducts activity follow-ups twice a week, and holds planning and retrospective ceremonies at the end of each 15-day cycle. Team members participated in the estimation activities both with and without the DevMetrics tool.

The experimental procedure was carried out in a real software development environment over the course of three consecutive full sprints with the selected development team from the company. DevMetrics was applied in different ways across the sprints to assess its effects at varying levels of maturity in using the tool, following a core principle of agile methodologies: continuous improvement.

Among the sprints, the first did not use the tool and served to establish a baseline of the team's estimation maturity. Planning was done using traditional empirical techniques (planning poker, team knowledge). In the second iteration, the tool was introduced and used to generate estimates for tickets integrated into Jira, but no changes were made to the team's existing process. Finally, in the third sprint, based on the analysis of previous results, process improvements were suggested to the team to generate more accurate estimates and promote standardization in their planning routines.

4.2 Results

Based on the case study described above, both quantitative and qualitative results were obtained that allow evaluating the effectiveness of DevMetrics in generating estimates and its acceptance by the development team. This section presents the accuracy indicators of the estimates generated across the three analyzed sprints, as well as user perceptions based on questionnaire responses at the end of the experiment.

During the case study, some limitations were encountered when applying the tool with the team. For instance, certain tasks did not have their actual time logged in Jira, reducing the usable sample of historical data.

Additionally, the agile team did not have a consolidated habit of writing standardized descriptions, which affected the effectiveness of the textual similarity calculations, as will be discussed below.

Considering the sequence of sprints described above, it was possible to evaluate not only the tool's effectiveness but also the direct impact of input data quality on its performance. Results show that in the first sprint (without DevMetrics), baseline values were established: MMRE of 0.418 and PRED(0.25) of 0.36.

In the second sprint, after adopting the tool, MMRE increased to 0.548 and PRED(0.25) dropped to 0.14. Since the tickets followed no documentation standards, the generated estimates ended up causing confusion for the development team.

It was observed that during the first sprint using DevMetrics, the team wrote only minimal descriptions for tasks. As a result, the estimate generation was misaligned with reality due to the small number of similar historical activities, which made it difficult for DevMetrics to perform meaningful similarity matching.

Thus, in the second sprint using DevMetrics, the team was guided to adopt some standard practices for refining tickets. These included labeling the title with "FE" (frontend) or "BE" (backend) at the start and writing descriptions using bullet points with short, direct sentences instead of long-form text.

Finally, in Sprint 3, after improving the task refinement process with the team, a significant improvement was observed. MMRE dropped to 0.285 and PRED(0.25) rose to 0.52, indicating that more than half of the tasks had an estimation error below 25%, as shown in Table 3.

Analyzing the metrics above, it is clear that the tool can support the team in generating estimates based on historical data, provided the team follows certain standards when documenting their activities.

Regarding the questionnaire results, the questions about system usefulness and willingness to reuse the tool in future sprints had average scores of 5.0 and 5.25 respectively, suggesting that participants found the tool useful and would be willing to use it again.

Table 3: Quantitative Results per Sprint

Sprint	Number of Activities	MMRE	PRED(0.25)
Sprint 1 (Without DevMetrics)	28	0.418	0.36
Sprint 2 (Inconsistent DevMetrics)	27	0.548	0.14
Sprint 3 (Standardized DevMetrics)	29	0.285	0.52

The team noted that the planning process improved even though it required additional time. The estimates generated by DevMetrics were considered more accurate.

Another highlight was the Jira integration, which reduced manual effort to synchronize activities. In the initial proposal, integration relied on direct access to the company's internal database, which required manually registering tasks and descriptions for estimation.

As for improvement points, survey results showed an average time-of-use score of 2.75, suggesting that the tool's use may have been considered time-consuming, potentially affecting work flow.

Another notable point was the usability and adaptability score of 3.75, interpreted as moderate to low, indicating some difficulty in adapting the tool to the team's workflow. This suggests a need for adjustments to make the integration process more seamless.

In conclusion, the metrics collected after applying the tool, along with the user feedback, indicate that DevMetrics is capable of generating effort estimates close to the actual time spent, provided the team follows certain documentation standards.

This reinforces the importance of input data quality and highlights the need to improve the setup and integration journey with team routines as a future direction for the tool's evolution.

5 Conclusion

This work aimed to build a support tool for software development teams to assist in sprint planning involving maintenance and system evolution activities.

This is a recurring challenge in companies that use agile methodologies and operate as software consultancies, a market that is continuously growing [11]. Furthermore, the cost of software maintenance represents a significant portion of the total budget of a development project [8].

Based on this context, the project adopted the premise described by [14], which suggests that historical data comparing estimated versus actual time spent can be used to generate new, more reliable estimates. The tool also aimed to recreate the estimation framework initially proposed by [16], known as EstimAi.

However, it was assumed from the outset that the final product should be a generic system that could be easily adopted by other companies and development teams, without requiring specific configurations for each organization.

Additionally, while textual similarity algorithms were preserved, the Sørensen-Dice method was replaced with the Jaccard method, which is more restrictive regarding differences between strings.

Statistical methods were also retained, combining linear regression for stable proportional relationships with exponential regression to handle abrupt variations caused by the sensitivity of the similarity algorithm. The final estimate was computed as the arithmetic mean of both models. Results were then normalized using the Fibonacci scale to better align with typical agile development scenarios.

To evaluate the effectiveness of the new tool, named DevMetrics, quantitative metrics were analyzed: estimated vs. actual time spent, using story points tracked in Jira. A survey was also applied to team members to assess the platform's impact during sprint planning ceremonies.

However, some limitations were observed regarding the additional time required to configure and learn the platform during sprint planning. It is expected that future iterations using the tool will reduce this time as the team overcomes the learning curve.

It is important to highlight that the tool has limitations related to input data, which must be standardized for the textual similarity algorithm to perform effectively. The team also needs to demonstrate maturity by consistently recording the actual effort spent on maintenance and feature development tasks to ensure the historical dataset is as complete as possible.

Another limitation identified was the number of participants in the study. It is believed that with a larger sample of users and teams using the platform, it would be possible to gather broader insights regarding improvement opportunities and the overall effectiveness of DevMetrics.

As for potential improvements to the open-source tool, simplifying the integration flow between Jira projects and DevMetrics could increase adoption among other companies and teams.

Another opportunity for refining estimates would be to add support for analyzing technical fields in addition to text, such as tags or labels commonly used in Jira projects. This would enable pre-filtering to focus estimations on tasks within the same label group.

Introducing the ICE index (Uncertainty, Complexity, and Effort) could also help standardize task descriptions. The index assigns low, medium, or high values to each of the three dimensions and could encourage development teams to reflect more critically on the work required, resulting in clearer task definitions.

A further improvement could involve presenting visual reports that compare system-generated estimates with actual values per sprint. This would provide useful inputs for agile teams during sprint retrospectives, helping to facilitate discussions on why estimation errors occurred.

Considering the limitations of traditional textual similarity algorithms, one possible evolution for the DevMetrics web tool is to incorporate natural language models (LLMs) into the process of identifying similar tickets. With a more robust model in place, it would be possible to compare not just titles and descriptions, but also acceptance criteria, allowing for a deeper semantic understanding of the content.

Such semantic interpretation is difficult to achieve with conceptual algorithms like Jaccard or Sørensen-Dice, which analyze word

structure rather than meaning. Additionally, LLMs could be used to automatically rewrite ticket descriptions, encouraging better documentation practices and improving input data quality.

Therefore, DevMetrics represents a promising step toward estimation processes grounded in historical evidence. It opens the door to more sophisticated developments involving AI, continuous refinement of historical data, and better alignment with team routines, ultimately optimizing the effort estimation process.

Future work includes benchmarking DevMetrics against classical and ML-based estimation methods, improving usability following Nielsen's heuristics, and releasing the system as an open-source project to encourage community adoption.

REFERENCES

[1] Atlassian. 2025. *Jira Software Cloud Documentation: Project Management, Issue Tracking, and REST API Reference.* https://support.atlassian.com/jira-software-cloud/ Disponível em: https://support.atlassian.com/jira-software-cloud/.

[2] Victor R. Basili, Gianluigi Caldiera, and Dieter H. Rombach. 1994. *Goal question metric paradigm.* Vol. 1. 6.

[3] Sai Mohan Reddy Chirra and Hassan Rezaorcid. 2019. A Survey on Software Cost Estimation Techniques. Journal of Software Engineering and Applications. https://doi.org/10.4236/jsea.2019.126014

[4] Fred D Davis, Richard P Bagozzi, and Paul R Warshaw. 1989. User acceptance of computer technology: a comparison of two theoretical models. *Management science* 35, 8 (1989), 982–1003.

[5] E. F. Franco, K. Hirama, and R. Rossi. 2018. Uma análise qualitativa-sistemática da relação entre o acúmulo de dívida técnica e a satisfação de usuários ao longo da operação de pacotes de software empresarial. (2018).

[6] Yara Maria Almeida FREIRE and Arnaldo Dias BELCHIOR. 2006. Estimativas de Manutenção de Software a partir de Casos de Uso. In *Simpósio Brasileiro de Qualidade de Software–III Workshop de Manutenção de Software Moderna, Vila Velha-ES.*

[7] Magne Jørgensen and Stein Grimstad. 2012. Does the Use of Fibonacci Numbers in Planning Poker Affect Effort Estimates?. In *Empirical Software Engineering and Measurement (ESEM).* –. Discusses how Fibonacci-based scales align with average estimation precision in software effort..

[8] Jussi Koskinen. 2015. Software maintenance costs. *University of Eastern Finland* (2015).

[9] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology* (1932).

[10] Marcos Alexandre Miguel, Marco Antônio Pereira Araújo, José Maria N. David, and Regina M. M. Braga. 2016. Um framework para apoiar estimativa de esforço em atividades de manutenção e evolução de software [A framework to support effort estimation on software maintenance and evolution activities]. In *SBSI.*

[11] Ministério da Ciência, Tecnologia e Inovações. 2022. Relatório MCTI – Software e Serviços de TIC no Brasil. https://www.gov.br/mcti/pt-br/acompanhe-o-mcti/noticias/2022/07/relatorio-do-mcti-aponta-que-industria-de-software-e-servicos-de-tic-cresceu-6-5-no-brasil-em-2021.

[12] Roger S. Pressman. 2016. *Engenharia de software: uma abordagem profissional.* (7ª ed.). McGraw Hill, Porto Alegre.

[13] Gonzalo Travieso, Renan Benatti, and Luciano da Fontoura Costa. 2024. An Analytical Approach to the Jaccard Similarity Index. *arXiv preprint arXiv:2410.16436* (October 2024). Available at: https://arxiv.org/abs/2410.16436.

[14] Carlos Eduardo VAZQUEZ, Guilherme Siqueira SIMÕES, and Renato Machado ALBERT. 2013. *Análise de Pontos de Função: Medição, estimativas e gerenciamento de projetos de software* (13 ed.). Érica, São Paulo.

[15] Tomas Vera, Sergio Ochoa, and Daniel Perovich. 2018. Survey of Software Development Effort Estimation Taxonomies. https://doi.org/10.13140/RG.2.2.14599.29601

[16] Kaliane Larissa Viesseli. 2021. Framework para estimar esforço de manutenção em um ambiente multi-equipe. *UTFPR-Journal* (2021).