An Approach to Automating User Story Management for Testing Process Optimization in Large Distributed Teams

Vitor Augusto Aguiar Trindade
Projetos, INDT - Instituto de Desenvolvimento Tecnológico
Manaus, Brazil
vitor.trindade@indt.org.br

Mateus Luiz de Oliveira Souza
Projetos, INDT - Instituto de Desenvolvimento Tecnológico
Manaus, Brazil
mateus.souza@indt.org.br

Hermann Jacques Hernani de Oliveira
Projetos, INDT - Instituto de Desenvolvimento Tecnológico
Manaus, Brazil
hermann.oliveira@indt.org.br

Luiz Henrique Aquino de Souza
Projetos, INDT - Instituto de Desenvolvimento Tecnológico
Manaus, Brazil
luiz.souza@indt.org.br

Bruno Cerdeira Pereira
Projetos, INDT - Instituto de Desenvolvimento Tecnológico
Manaus, Brazil
bruno.cerdeira@indt.org.br

José Carlos Rangel do Nascimento
ICOMP, UFAM - Universidade Federal do Amazonas
Manaus, Brazil
jcrn@icomp.ufam.edu.br

Eliane Collins
Projetos, INDT - Instituto de Desenvolvimento Tecnológico
Manaus, Brazil
eliane.collins@indt.org.br

ABSTRACT

Efficiently managing user stories during the testing process is crucial in large, distributed test teams to ensure timely and high-quality project completion. Manual control of tasks, from various software testing activities to extracting specific quality metrics and maintaining test processes, is time-consuming and error-prone. This paper proposes an approach to automate story creation for the Jira tool to address these challenges. The proposal approach aims to optimize the time spent on story creation, promote efficiency, and increase the success rate of story creation from 73.43% to 99.66% in the story management process. The script developed establishes a secure connection with Jira, imports data from spreadsheets, populates these into Jira, and creates or updates test tasks. Additionally, JavaScript (JS) code automates data transfer from task creation to planning. This automation significantly reduces manual effort and minimizes human errors, due to the improvement in the success rate to 98.14%, and increases the productivity of the software testing team by 15.19%.

KEYWORDS

Software Process Improvement, JIRA Automation, User Story, Software Testing, Software Project Management, Data Analysis

1 Introduction

In the dynamic and competitive global software market, efficient test processes are necessary to ensure the quality of new products and software [12]. Task management in large, distributed test teams is essential to ensure process execution and completeness, and metrics are collected to identify problems in process execution progress that could delay delivery.

Jira is a highly used tool by software development and testing teams. According to Parabol [11], Jira is the most popular software used for Agile management among development teams, highlighting its widespread adoption in the industry [10]. Additionally, a report from Zippia [17] confirms that Jira's Agile features make it a favorite among Agile teams. In Jira, stories are a type of issue that represents individual units of work within a larger project or sprint. They are commonly treated as tasks that must be completed to complete the sprint.

Given this context, effective story management on Jira is particularly critical in large scale software testing teams, where stories need to be created, updated, and monitored regularly by globally distributed teams. Traditional methods of story management are often related to manual processes that consume time and are prone to errors. This situation impacts not only the efficiency of the team but also increases the risk of delays and inconsistencies in project schedules, or even metric mistakes. In that respect, manual story management involves numerous repetitive steps, including story creation, story assignment, monitoring, etc [2]. Each of these steps is susceptible to errors, such as incorrect data entry, lack of information or fields, bridging potential problems, or data inconsistency in large software projects. Additionally, the time spent on these manual processes causes the focus to change away from valuable resources to more critical analytical and strategic tasks.

Moreover, as teams grow and demands become bigger, the volume of stories increases proportionally, increasing the inefficiencies and errors associated with manual story management, as discussed in the study on the impact of team size on productivity in software development (Rodriguez et al., 2011) [13]. This situation highlights the need for an automated solution to expedite the creation and maintenance of stories.

The proposal approach presented in this study was developed to support the test process in large teams, offering an automated solution for story management and maintenance on Jira. This application assists the test activities, integrating with other tools for a more cohesive workflow [3]. The main goal is to reduce the time spent on testing, planning, and control by increasing the accuracy of the information entered, thereby supporting the improvement of the testing team's process.

This integrated approach can optimize story creation time and promote accuracy and consistency of quality metrics regarding team effort. Reducing these manual processes in the industry case study obtained a 98% success rate and frees the test team to focus on higher-value activities, such as requirements analysis for new functionalities and creation and update of test cases, ensuring more than 23.87% efficiency. The script implementation represents a trend of benefits to the software testing team and potentially as a model for other similar teams and organizations.

This paper makes the following contributions:

- The test process improvement through automation of story creation using the integration of popular tools, like Jira and Jenkins.
- A case study evaluation demonstrating the script to support the test process in a similar industry context.

This paper is organized as follows: Section 2 provides the background. Section 3 offers the related works. Section 4 details the methodology step-by-step and discusses the application approach to the story management, with flowcharts and technical discussions. Section 5 shows the case study evaluation. Section 6 presents the results analysis. Finally, Section 7 offers future research directions and conclusions.

2 Background

This section presents the technical background for the software testing, mobile application testing, and Scrum processes. According to (MYERS, 2004) [9], software testing is a process, or a series of processes, designed to ensure that computer code does what it was intended to do and does not do anything unintended. Software testing has become essential for companies to ensure product quality, regardless of the development methodology.

The test process generally consists of the following activities throughout the software development: planning and control, analysis and design of test cases, implementation and execution of tests, test reporting, and test closure (Figure 1). (VEENENDAAL; GRAHAM; BLACK, 2008) [4].

Figure 1: Software Testing Process

Test managers or leaders work with customers in the test planning phase to establish the test objectives. Then, a test plan document is generated to describe strategies, test scope, resources, methods, test techniques, completion criteria, and a schedule of activities during the project. The test plan can be based on the standard format established by IEEE 829-2008 [1].

The test manager is responsible for the test team and ensures the control of the tasks (stories) established in the testing process.

In the analysis and design of test cases, the test objectives are turned into tangible test cases and test conditions. Test cases must be complete, reproducible, and independent. A test case contains identification, prerequisites, steps, and expected output. In addition, it must be designed to discover unexpected software failures. In this activity, test cases are created in a document called Test Case Specification (VEENENDAAL; GRAHAM; BLACK, 2008) [4].

The environment must be configured to allow coding test scripts in the implementation phase. Then, the test execution must be run, and the results must be recorded in a Test Execution Report and communicated to the project team. Once the test activities meet the exit criteria in the test plan, the activities, such as results, logs, and test documents related to the project, are archived and used as a reference for future projects.

Some type of testing is essential to understand the process of execution cycles:

- Regression Test: Repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered due to the changes in the software being tested or in other related or unrelated software components. (Software Testing Material, n.d.) [14].
- Sanity Test: This test is performed after receiving a software build with minor changes in the code. The testing aims to verify the "rationality" before proceeding with more rigorous testing. Sanity testing exercises only the particular component of the entire system (Guru99, n.d.) [6].
- Exploratory Test: 'Exploratory' is an approach where testers actively explore the software to identify issues and assess user experience without relying on predefined test cases. (Browse stack, n.d.) [5].

2.1 Scrum

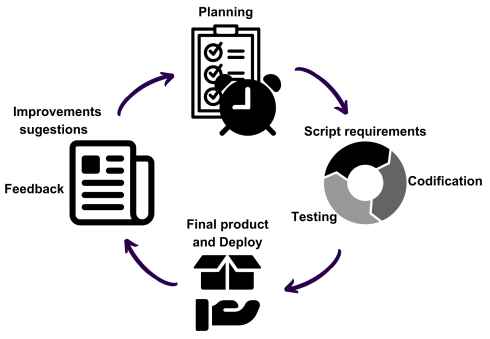

Figure 2: Methodology flowchart

Sprints: The development was divided into weekly sprints. Each sprint began with a Sprint Planning meeting, where the team selected the points to be worked on and defined the sprint goals.

Daily: Daily meetings to share progress, identify impediments, and align activities.

2.1.1 Iterative and Incremental Methodology. By integrating the iterative and incremental methodology within Scrum, the team can ensure a structured and flexible approach to development. This combination allows continuous delivery of valuable increments, regular feedback, and ongoing improvement, ensuring that the final product meets user needs and quality standards.

3 Related Works

In the study of Pontes and Rebelo (2021), the tool Task Verifier was developed to be capable of automatically querying the Jira system for the tasks that have been incorrectly filled according to some search criteria defined in JQL and of notifying their respective assignee to tell precisely which field to adjust. The project manager made the same effort, but in an immensely lower time and with the additional functionality of presenting a web link to a page containing instructions on filling the incorrect field. This tool directly notifies the assignees whenever it is executed (possibly multiple times per month), telling them what inconsistencies were identified in their tasks and offering brief instructions on how to fix them, promoting the improvement of the test process.

In Li et al. (2015) [8], a suitable test model and FPGA (Field Programmable Gate Arrays) software testing process management system are proposed, according to the FPGA test situation and the features of the FPGA design. It is a standardized, orderly, systematic, engineering-oriented, task-oriented document and related management tools. The testing activities can provide correct guidance, organization, and implementation; they can also be used for continuous improvement in all stages of the testing process, work quality, and utility. Early discovery and closure of the defects of the FPGA in the development process improve communication efficiency between the FPGA designer and tester, and ultimately ensure the quality of FPGA products, improving customer satisfaction.

The work of Unudulmaz et al. (2020) [16] applies Scrum practice to the Test Maturity Model Integration (TMMI) to reduce the documentation and improve test process quality. Agile and test practices map the "TMMI Level 2 - Managed" step. Test Policy and Strategy, Test Planning, Test Monitoring and Control, Test Execution, and Test Environment areas are discussed to match Scrum practices and test practices (Risk Based Testing, Use Case Testing). Finally, the TMMI coverage percentage was given, and improvement areas were defined. TMMI Level 4 metrics were used for a development project, and the results will be shown. This model can also be adapted to different areas that use agile and Scrum practices.

These related studies show the importance of the input to evaluate the need to improve test process. The novelty of the proposed approach lies in automating the creation and updating of tasks (Stories) directly in the test planning phase, significantly reducing manual effort and preventing inconsistencies before they occur. Existing studies do not focus on story creation and operate in later stages. In contrast, the proposed approach provides an automation that generates a visual planning spreadsheet and creates multiple stories based on a single input spreadsheet. This study contributes to this scenario by focusing on the effort of creation of tasks (Stories) in the phase of test planning, saving time for more valuable activities in testing process.

4 Methodology

This section details the methodology for developing the proposal approach to optimizing story creation and management within software testing teams.

The methodology integrates automation tools, scripting, and iterative development processes to address the inefficiencies associated with manual story management. This approach aims to streamline the workflow, reduce human errors, and enhance productivity. The approach outlines the step-by-step implementation of the methodology, the technologies used, and the script workflow.

4.1 Development Process

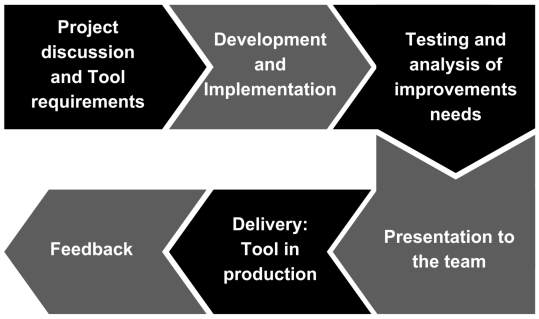

Figure 3: Development Process of the script

Figure 3 illustrates the development lifecycle of the story creation script. The process begins with the requirements analysis phase, which involves studying user needs to define system, hardware, or software requirements [1]. This phase involves initial discussions and requirements analysis, essential for aligning the team's understanding and expectations. Next, the development and implementation phase involves coding and integration of the script, where developers translate requirements into functional software.

Following development, the testing and analysis of the improvement needs phase are conducted. This phase involves identifying bugs or areas for enhancement using Unit Testing, a process that tests program components, such as methods or object classes. Additionally, Component Tests are performed, as interface errors in composite components may not be detectable through testing on individual objects [15]. Once testing is completed, the Presentation to the Team phase occurs, where the script is demonstrated to the team for feedback and approval.

Upon successful presentation, the delivery script in the Production phase follows, marking the deployment of the script into the production environment. Finally, the Feedback phase involves collecting and analyzing user feedback to further refine and improve the script. This continuous feedback loop helps to maintain the script's effectiveness and relevance.

4.2 Proposal Approach

The test leaders of the company populate a test planning spreadsheet to demand the necessary working hours of the 17 team members for sprint activities. Then, the stories are cloned into the Jira tool from similar stories from the previous sprint, requiring time to edit and update stories, to include the specific labels and software information required by the client, which demands time to adjust mandatory fields and adjust team members' hours according to the sprint plan. It recurrently generated human errors in cloning an activity and not adjusting the mandatory fields for the sprint, causing rework and stops for adjustments to deliver accurate metrics and information to the client and stakeholders. Figure 4 shows an example of the planning spreadsheet used in the company process.

SMOKE TEST						
Task Name	**Priority**	**Description**	**Total Hours**	**Task Owner**	**Task Participants**	**Hours**
Monday plan	1	Execution of test plan	7	James	James	7
Tuesday plan	2	Execution of test plan	10.5	Paul	Paul	3.5
					Matthew	7
Wednesday plan	1	Execution of test plan	7	Chris	Chris	7
Thursday plan	1	Execution of test plan	14	Victor	John	7
					Victor	7
Friday plan	1	Execution of test plan	7	Bruce	Bruce	7

REGRESSION TEST						
Task Name	**Priority**	**Description**	**Total Hours**	**Task Owner**	**Task Participants**	**Hours**
First test plan	3	Analysis of test plan	30.5	Anne	Anne	30.5
Test execution	1	Start execution	11	William	WIllian	11

Figure 4: Development Process of the script

This proposed approach aims to optimize time and enhance the efficiency of story management. This is achieved by implementing a JS code that automates the management of two key spreadsheets: the stories Spreadsheet, which is used to populate data, and the Planning Spreadsheet, which is used to present the activities in a more readable human way during the Scrum Sprint Planning meeting. The main Python script populates the data in Jira and creates or updates the stories. The following sections outline the detailed workflow of the JS script and the main program, accompanied by their respective flowcharts.

Figure 5: Objective Flow

The script establishes a secure connection with Jira, enabling access and manipulation of story data. Subsequently, data from a spreadsheet is imported into the script, which inserts the information into the corresponding fields in Jira. This process facilitates the creation of new stories or updates to existing ones as necessary (Figure 5).

4.3 Technologies Used

The development of automation logic applies to Python as its primary programming language. Data analysis and manipulation are facilitated through the use of Pandas, a software library written for Python. The coding process uses Visual Studio Code, an integrated development environment (IDE). Story integration and management are achieved through Jira, a tool usually used for this purpose. Jenkins automates the script's execution, which schedules the job to run at programmed intervals. Data collection and visualization are done in real-time using Google Spreadsheets, while JS scripts automate data transfer between spreadsheets.

4.4 Script Workflow

The story creation process involves populating a spreadsheet with the necessary data. A JS code is automatically triggered after completing the story creation spreadsheet (Figure 6). This script starts by identifying the target Sheet document using its unique ID. It then extracts relevant story data, focusing on summarized, objective information and issue-type (task or sub-task), and transfers this data from the story creator spreadsheet to the main planning spreadsheet (Figure 6). A sub-task issue-type is especially required when the same work activity needs to be divided among multiple people working on it during the same Sprint window. Therefore, the sub-task can handle the complexity of tracking separate ongoing progress and allocating different hours to each sub-task. In the planning spreadsheet, team members define Story participants and the effort required.

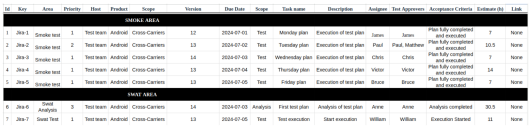

Figure 6: User story creation spreadsheet filled

During the planning meeting, the script checks if the story already exists in Jira using the story key (if provided in the sheet). After the Scrum Sprint Planning is completed, the story creation sheet is updated with the latest changes from the planning meeting, ensuring that any new story allocations or modifications are accurately reflected. This marks the conclusion of the JS process. Subsequently, a Jenkins job executes the main script, which imports data from the stories spreadsheet, populates Jira fields accordingly, and creates or updates stories as needed, thus completing the process and displaying completion messages or errors, if any. This workflow is expressed in Figure 7.

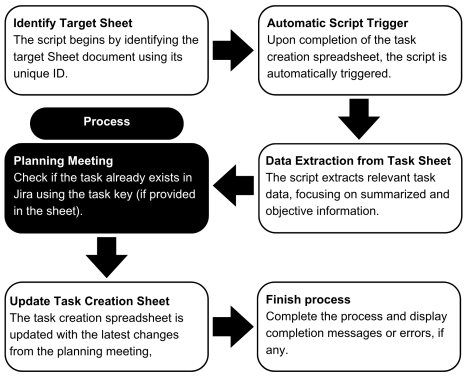

Figure 7: Stories automation flow

4.5 Backend Python

The script requires specific arguments to initiate an execution. It will be necessary to provide a Jira service account, password, board ID (a numerical reference to the Jira board), and the key of the spreadsheet where each column represents a field to be filled in and each row a complete set of fields to create one story. Those arguments are passed to a Jenkins job mentioned in section 4.4, and the executions can be triggered periodically.

As shown in Figure 7, the first step is to authenticate with valid credentials whose privileges allow the creation of stories on Jira. The subsequent steps include reading and processing the spreadsheet data using pandas, cleaning data, casting some columns to specific types, ensuring their integrity, and preventing bugs. A check is performed for each row on the "Key" column: if the field is empty, a new story is created; otherwise, an existing story is updated. At each iteration, a Python object is created containing all relevant fields for issue creation, which is then passed to the Jira API's create*¡ssue* method to create or update [7] the issue. Upon completion, each field in the "Key" column of the spreadsheet is updated with the new ID of the created issue.

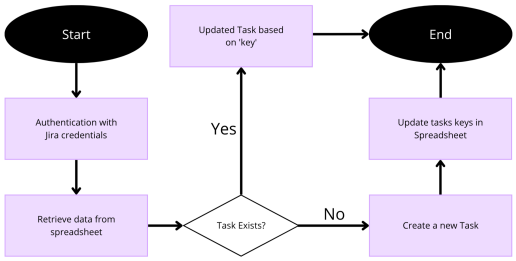

Figure 8: Stories automation flow

4.6 Challenges

During the implementation process, several technical challenges were encountered, particularly in automation and system integration. The automated creation of tasks in Jira from worksheet data required the development of a validation script to ensure data consistency. To address rate-limit issues (HTTP 429), controlled intervals were introduced between the creation of Stories, and periodic execution scheduling was implemented through Jenkins pipelines, providing greater predictability and operational control. Another major challenge involved the creation of sub-tasks, which creation was not fully supported by the Jira API, as the current API calls are not able to update Jira custom fields. This limitation was overcome using Playwright for graphical interface automation and content scraping, executed on a server with GUI support and automatic authentication via a service account, ensuring process continuity without manual intervention.

5 Case Study Evaluation

In this section, the results obtained from the implementation of the proposed solution in a real-world scenario are presented. The evaluation was conducted within a software development and validation project for Android mobile devices, involving a testing team of 17 professionals, of which 12 are responsible for creating approximately 85 stories each sprint. This is a common practice in industries that employ Kanban-based workflows, where Stories are created to manage tasks. Each Story typically includes the allocated time for the activity, the devices, plataforms, and/or operating systems used, and the start and end times of the task.

5.1 Team Environment

The team operates in a dynamic and collaborative environment, using a variety of scripts and technologies to conduct software testing and manage stories. The key aspects of the environment include:

Machines Used: The team utilizes high-performance workstations and 64-bit laptops tailored for software development and testing, featuring specifications such as the Intel Core i7-13700K 2.8GHz processor, DDR4 16GB RAM, and SSD storage ranging from 512GB to 1TB. These machines operate on both Windows 11 and Ubuntu 22.04 operating systems.

Team methodology The team operates under the Scrum methodology, conducting weekly sprints where stories are planned and executed in seven days. Each sprint consists of stories planned during the Scrum Sprint Planning session. Daily meetings are held to synchronize efforts and address possible impediments, ensuring continuous progress towards the sprint goals.

The study was executed as follows: Metrics for time, errors, efficiency, accuracy, and story creation success rate were collected. After gathering this data, it was organized into tables, and comparative charts were developed for each evaluated metric. This allowed for a detailed study of the results, demonstrating trends in improvements. Based on these trends, the results and metrics were evaluated and analyzed to gain insights into the effectiveness of the automated story creation process.

5.2 Collected Data

To evaluate the script's impact, a dataset containing the story creation history for 21 sprints was compiled - seven sprints before the script's implementation (Table 1), seven sprints with the team in an adaptation period (Table 2), and seven sprints after continuous use of the tool (Table 3). The key metrics used for evaluation included the number of stories created, execution time, success rate, and the number of errors in stories. The number of stories created is the

number of story-type issues created per sprint. Execution time is the average time for the script to execute the story creation or the manual effort to create it. Success rate is the proportion of stories created successfully in relation to the total number of stories attempted. The number of errors in stories is the sum of the fields of all stories that are incorrectly filled or missing fields that need to be updated.

Table 1: Metrics before the story management approach

Sprint Week (SW)	Stories Created	Creation Time	Success Rate	Number of Errors
SW 01	45	2.23h	80%	9
SW 02	38	1.82h	84.21%	6
SW 03	39	1.82h	66.67%	13
SW 04	41	2h	82.83%	7
SW 05	42	1.92h	73.81%	11
SW 06	41	1.90h	68.29%	13
SW 07	39	1.95h	61.54%	15

Table 1 shows a high number of errors, a low success rate, and an extended time required to create stories, approximately two hours per sprint. Note that these metrics are related to a process that was manually performed. Additionally, an unspecified time in that table was spent reviewing and correcting errors in the stories, which took around four and a half minutes per task, totaling an average of three hours per sprint. This effort, however, was no longer necessary in the data collected after the script, even during the adaptation period, as the creation process became automated, eliminating the possibility of missing or incorrectly filled fields.

Table 2: Metrics of the Script

Sprint Week (SW)	Stories Created	Creation Time	Success Rate	Number of Errors
SW 08	61	0.33h	95.08%	3
SW 09	60	0.33h	96.67%	2
SW 10	64	0.68h	98.44%	1
SW 11	59	0.32h	100%	0
SW 12	61	0.33h	100%	0
SW 13	63	0.48h	96.83%	2
SW 14	59	0.32h	100%	0

The number of previously created stories was smaller than after the script was implemented, as the previous process used sub-tasks. The process was adjusted so that the script could read and create activities that were previously done in sub-tasks, thus better measuring the effort determined in each activity. This promoted greater efficiency, as shown in Figure 8. It is worth noting that these results were obtained during an adaptation period, and the script already demonstrated considerable improvements in productivity and precision.

Table 3: Script Metrics after full integration and continuous use

Sprint Week (SW)	Stories Created	Creation Time	Success Rate	Number of Errors
SW 15	52	0.28h	100%	0
SW 16	65	0.37h	100%	0
SW 17	75	0.43h	100%	0
SW 18	80	0.50h	98.75%	1
SW 19	88	0.68h	98.86%	1
SW 20	83	0.57h	100%	0
SW 21	82	0.51h	100%	0

With the tool already integrated into the team's daily process, some process changes occurred, and more stories per sprint needed to be created. As a result, there was a 50% increase in the number of stories, rising from an average of 60 at the start of the script's use to 85. Even with this increase, the script maintained nearly 100% of precision, with minimal error rates, most of which were due to network issues. The improvements in efficiency and accuracy can be seen in Table 3.

5.3 Efficiency Analysis

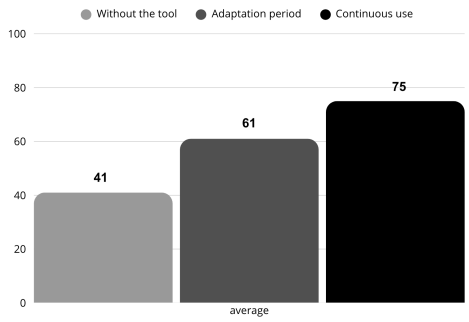

Figure 9: Efficiency Chart

Figure 9 illustrates the evolution of the average number of stories created per sprint across three different stages of tool adoption. Initially, without using the script, the team maintained an average of 41 stories per sprint, reflecting a fully manual process that was time-consuming and prone to inconsistencies.

During the adaptation period, a significant increase in productivity was observed, with the average rising to 69 stories per sprint. This improvement can be attributed to the reduction in time required for story creation and the increased efficiency introduced by partial automation.

Finally, with the continuous use of the script, the process, which required no manual time for story creation, resulted in the average number of stories created per sprint reaching 75, highlighting the effectiveness and impact of the tool in optimizing the planning process. These figures represent averages, but in some sprints, up to 88 stories were created - more than double the average before the tool's implementation - demonstrating the substantial productivity gains enabled by the automation.

6 Results Obtained (industry scale cases)

This section presents the results obtained from the implementation of the proposed story automation script in a test team scenario. The analysis was carried out over multiple sprints, both before and after script implementation, to measure its impact on story creation efficiency (Figure 8), precision (Figure 10), and optimizing the team's time (Figure 9). The metrics used for the evaluation include the number of stories created, the execution time, and the success rate of the story creation.

6.1 Metrics Analysis

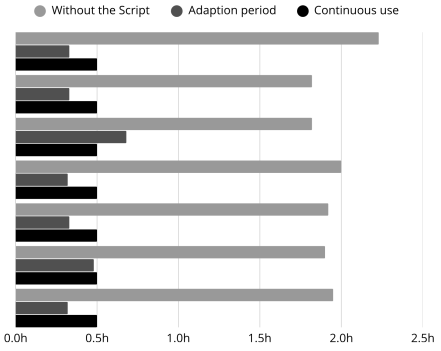

Figure 10: Time Chart

Without the Script: As seen in the chart above, the story creation process took around 2 hours per sprint to create 40 stories. This process had a high rate of human errors, which affected the team's metrics and cost significant time in the current sprint.

Adaptation period: The automation of the story management process allowed a significant increase in story correctness and a considerable reduction in the time spent. These new executions, which worked with more than 60 stories per sprint, lasted about 20 minutes, and in the final executions, the human error rate, according to the collected metrics, was 0%, which ensured consistency and precision. Despite this execution time of 0.3 hours, the time spent in the current sprint by the team was zero, as the Jenkins Job runs the script even before the start of the sprint. This automation improved a 80% reduction in the time to create the story; however, from the perspective of human effort, the required time became effectively zero.

Continuous use of the tool: After the adaptation period, since no human time was required for story creation, it became possible to increase the number of stories created, increasing from an average of 60 to 75, and in the last sprints, reaching nearly 90. In the chart in Figure 10, an increase in the number of stories results in a creation time of approximately 0.5 hours; however, this effort corresponds only to the automated job, while human time remains zero.

6.2 Impact on Productivity

The script reduced the manual effort of the software testing team to create its stories, allowing team members to focus on more strategic and analytical activities, collaborating with overall team performance and productivity.

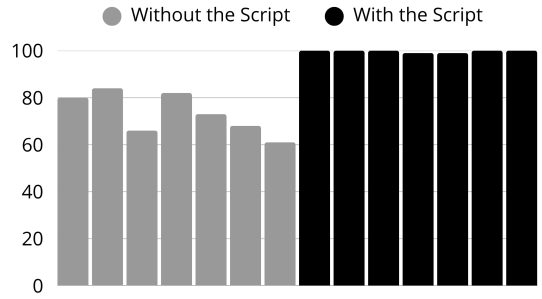

Figure 11: Accuracy Chart

Improved accuracy and consistency in story details ensured that project schedules and metrics were more reliable, reducing the risk of delays and inconsistencies.

The results demonstrate a trend that the proposed story automation script provides substantial benefits in terms of efficiency, accuracy, and productivity, according to the collected metrics, while not causing any bottlenecks in later stages, since all of them are automated. This success suggests that similar automation approaches could be beneficial for other teams and organizations facing similar challenges in story management.

7 Conclusion

The main contribution of this work, compared to Jira's native automations and marketplace solutions, lies in its specialized and measurable approach tailored to the software testing domain within larger distributed teams. Unlike existing alternatives, which are often generic and not fully aligned with testing-specific requirements, our proposal enables direct integration between spreadsheets and Jira for creating and updating stories, ensuring greater consistency and reducing manual effort. In addition, the use of JavaScript to automate the transition of information from the creation to the planning phase, together with Playwright to overcome Jira's API limitation in populating custom fields, establishes a seamless workflow that is not available in other existing tools.

This paper has introduced a Python-based approach for automating story management in Jira, demonstrating its impact on efficiency and accuracy within a software testing process in large teams. Before implementing the script, the story creation process took around 2 hours per sprint to generate around 40 stories, often plagued by human errors that disrupted team metrics and consumed valuable sprint time.

Upon integrating the automation script, improvements were observed, as it enabled the creation of more than 80 stories per sprint in 30 minutes on average, with no human error in the final executions. This enhanced precision and consistency eliminated the need for manual corrections, contributing to a streamlined workflow that improved the project schedule and operational metrics.

The metrics collected throughout the implementation phase underscored the script's impact on productivity. By reducing manual effort, team members were freed to focus on strategic and analytical

tasks, thereby optimizing overall team performance. The automation script also demonstrated a trend to mitigate risks associated with project delays and inconsistencies.

The approach was designed to be used by other software testing teams within the same organization, minimizing the need for adjustments. Therefore, it has general applicability, as automation significantly reduces manual work when creating stories from the Scrum planning; in smaller teams, the benefit can be even greater because it frees up effort from story creation. Furthermore, the approach can be adapted to other toolchains with minor adjustments in the integration layer, which is responsible for connecting the automation scripts with tools such as Jira. However, future research is exploring adapting the approach to accommodate different project complexity and integration requirements. By expanding functionalities and refining automation parameters, organizations can further optimize their software testing processes, driving continuous improvements in efficiency, accuracy, and team productivity.

ACKNOWLEDGMENTS

This research was done during the year 2024 and was funded as provided for in arts. 21 and 22 of decree no. 10,521/2020, under the terms of Federal Law no. 8,387/1991, through agreement no. 003/2021, signed between INDT - Instituto de Desenvolvimento Tecnológico, Flextronics da Amazônia Ltda. and Motorola Mobility Comércio de Produtos Eletrônicos Ltda.

REFERENCES

[1] 1990. IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990* (1990), 1–84. doi:10.1109/IEEESTD.1990.101064
[2] Atlassian. 2024. Create issues and subtasks. https://support.atlassian.com/jira-work-management/docs/create-issues-and-subtasks/. Accessed: 17-Jul-2024.
[3] Atlassian. 2024. Jira Software Documentation. https://www.atlassian.com/software/jira. Accessed: 23-Jul-2025.
[4] G. Bath and E. Van Veenendaal. 2013. *Improving the Test Process: Implementing Improvement and Change - A Study Guide for the ISTQB Expert Level Module*. Rocky Nook. https://books.google.com.br/books?id=oT-4BAAAQBAJ
[5] BrowserStack. 2025. Exploratory Testing Guide: Definition, How To Perform It, & When To Use It. https://www.browserstack.com/guide/exploratory-testing. Accessed: 23-Jul-2025.
[6] Guru99. 2025. Smoke Testing Vs Sanity Testing: Differences & Comparison. https://www.guru99.com/smoke-sanity-testing.html. Accessed: 23-Jul-2025.
[7] Jira Documentation. 2024. Jira REST API. https://jira.readthedocs.io/api.html. Accessed: 15-Jul-2024.
[8] Wang Li and Zhou Hao. 2015. FPGA software testing process management. In *2015 IEEE International Conference on Grey Systems and Intelligent Services (GSIS)*. 600–603. doi:10.1109/GSIS.2015.7301927
[9] G.J. Myers, C. Sandler, and T. Badgett. 2011. *The Art of Software Testing*. Wiley. https://books.google.com.br/books?id=GjyEFPkMCwcC
[10] Parabol. 2023. The Most Popular Agile Software Tools in 2023. https://www.parabol.co/blog/agile-tools/. Accessed: 17-Jul-2024.
[11] Parabol. n.d.. Parabol. https://www.parabol.co/. Accessed: 17-Jul-2024.
[12] Eliane Figueiredo Collins Ribeiro. 2022. *DeepRLGUIMAT: Deep Reinforcement Learning-based GUI Mobile Application Testing Approach*. Tese (Doutorado em Ciências de Computação e Matemática Computacional). Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos. doi:10.11606/T.55.2022.tde-03062022-162453 Acesso em: 23 jul. 2025.
[13] Daniel Rodriguez, M. Sicilia, Elena Barriocanal, and Rachel Harrison. 2012. Empirical Findings on Team Size and Productivity in Software Development. *Journal of Systems and Software - JSS* 85 (03 2012). doi:10.1016/j.jss.2011.09.009
[14] Software Testing Material. 2025. Regression Testing. https://www.softwaretestingmaterial.com/regression-testing/. Accessed: 23-Jul-2025.
[15] I. Sommerville. 2011. *Engenharia de software*. Pearson Prentice Hall. https://books.google.com.br/books?id=H4u5ygAACAAJ
[16] Ahmet Unudulmaz and Oya Kalıpsız. 2020. TMMI Integration with Agile and Test Process. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering* (Trondheim, Norway) *(EASE '20)*. Association for Computing Machinery, New York, NY, USA, 375–378. doi:10.1145/3383219.3386124
[17] Zippia. 2023. Jira Agile Market Share and Competitor Report. https://www.zippia.com/advice/jira-agile-market-share/. Accessed: 17-Jul-2024.