Jana Batskihh

DEVOPS APPROACH IN SOFTWARE DEVELOPMENT USING ATLASSIAN JIRA SOFTWARE

Bachelor's thesis

Information Technology

Bachelor of Engineering

2023

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering
Author(s)	Jana Batskihh
Thesis title	DevOps Approach in Software Development Using Atlassian Jira Software
Commissioned by	Playtech Estonia OÜ
Year	2023
Pages	82 pages, 0 pages of appendices
Supervisor(s)	Timo Mynttinen

ABSTRACT

This thesis explores the ways in which automation and optimization of everyday tasks, leveraging the DevOps philosophy and Atlassian Jira Software, can significantly enhance users' productivity and effectiveness in their daily routines. The theoretical part delves into the principles of DevOps, providing a comprehensive and detailed understanding of the underlying concepts. The primary objective of this thesis was to automate and streamline processes, ultimately accelerating the overall software development process within the organization.

This study employed a range of plugins, including both native and third-party ones, to achieve the objectives of time, resource, and budget savings. It comprehensively elucidated diverse approaches to enhancing teams' daily working routines. Furthermore, a survey was conducted to assess the significance of the implemented automation through plugins in users' day-to-day work.

The study demonstrated that teams responded positively to the implementation of various automation rules and improvements. The changes made to the environment served as a benchmark for other projects within the company. The commissioner greatly benefited from the work outlined in this study, and furthermore, companies from diverse sectors can also find these results valuable for their own teams and projects.

Keywords: devops, jira, atlassian, automation

CONTENTS

1 INTRODUCTION 4

2 THEORETICAL PART 4

2.1 What is DevOps? 5

2.2 Roles in DevOps 7

2.3 DevOps frameworks 9

2.4 DevOps tools 15

2.5 Relationship to other approaches 20

2.6 Atlassian systems 23

2.6.1 Jira 24

2.6.2 Confluence 26

2.6.3 Others (Bitbucket, Opsgenie, Crucible, Fisheye) 26

3 PRACTICAL PART 28

3.1 Commissioner overview 28

3.2 Tools and plugins 29

3.3 Test project setup 29

3.3.1 Email and Microsoft Teams notifications about creating new issue 31

3.3.2 Automatic sub-task creation 41

3.3.3 Pre-set description of an issue 50

3.3.4 Auto-assignment to QA user 54

3.3.5 Restricting transitions: condition blocking from closing issue 61

3.3.6 Automatic closure of the parent issue when sub-tasks are completed 65

3.3.7 Issue Matrix: detailed issue view under the Epic 71

3.4 Review of results 75

4 CONCLUSION 76

REFERENCES 79

1 INTRODUCTION

The world we live in today is fast-paced and constantly evolving. Businesses must keep up with the changing demands of their customers and the competitive market. Information technology tools are an integral part of nearly every aspect of our lives, from the mobile applications we use for food delivery and transportation, to the means by which we connect with friends, family, and government services. As technology rapidly evolves, businesses must adapt to provide the best possible services to their customers.

The need for the DevOps approach arises precisely in this context. DevOps refers to a software development method that prioritizes collaboration, communication, and automation between software development and IT operations teams in order to enhance the speed, efficiency, and quality of software delivery. Given the competitive nature of the industry, businesses must strive to develop their products as efficiently as possible, with minimal delays and maximum efficiency. By effectively implementing DevOps philosophy and tools, businesses can reap the benefits of this approach and thrive.

Although DevOps encompasses many different roles, this thesis concentrates on the implementation of DevOps through tools from an administrator's perspective, with a particular emphasis on Jira. The central goal of this thesis is to illustrate the DevOps approach by simulating realistic scenarios, drawing on my practical experience as an Atlassian services administrator.

2 THEORETICAL PART

This chapter provides an overview of the DevOps philosophy, including its purpose, team roles, and responsibilities. Additionally, it examines DevOps frameworks and tools, with a particular emphasis on Atlassian products, especially Jira, which will be utilized in the practical study.

2.1 What is DevOps?

The DevOps movement emerged in approximately 2007-2008 in response to concerns raised by the software development and IT operations communities regarding the conventional software development model. In this model, developers and operations teams worked in isolation from each other, with developers responsible for writing code, while operations deployed and provided support for the code. DevOps, a term that blends the words "development" and "operations," represents the effort to unify these two fields into a seamless and uninterrupted process. (Atlassian, 2023)

A DevOps team comprises of developers and IT operations working together in a collaborative manner during the product lifecycle, aiming to enhance the velocity and excellence of software deployment. It constitutes a fresh mode of operation, signifying a cultural transformation, that holds significant consequences for both teams and the organizations they are associated with. (Atlassian, 2023)

In a DevOps model, development and operations teams no longer operate in isolation. At times, these teams blend into a single unit where the engineers operate across the entire application lifecycle - from development and testing to deployment and operations - with a varied range of interdisciplinary skills.

DevOps teams utilize tools for automating and accelerating processes, which leads to increased dependability. A DevOps toolchain helps teams address vital DevOps essentials like continuous integration, continuous delivery, automation, and collaboration.

According to the 2020 DevOps Trends survey conducted by Atlassian, almost all the participants (99 percent) reported that DevOps had a favorable effect on their organization. DevOps offers several advantages such as quicker and smoother releases, improved team productivity, enhanced security, better-quality products, and ultimately increased satisfaction among teams and customers. (Atlassian, 2020)

The following are the benefits of DevOps usage (Atlassian, 2023):

Speed

DevOps teams tend to release their products more often, and with better quality and stability compared to non-DevOps teams. According to the DORA 2019 State of DevOps report (Forsgren, 2019), highly efficient teams deploy their deliverables 208 times more frequently and 106 times faster than low-performing teams. By employing continuous delivery, teams can automate the process of building, testing, and delivering software with the help of various tools.

Improved collaboration

DevOps relies on a collaborative culture between development and operations teams, where they work together and share responsibilities. This approach enhances the efficiency of the teams and reduces the time required for work handoffs and developing code that suits the environment in which it operates.

Rapid deployment

DevOps teams can enhance their products rapidly by increasing the frequency and speed of their releases. They can gain a competitive edge by quickly introducing new features and fixing any issues in their products.

Quality and reliability

Continuous integration and continuous delivery practices guarantee that any changes made to software are operational and secure, which leads to better product quality. Real-time monitoring enables teams to stay updated on performance.

Security

DevSecOps is an integrated and active aspect of the development process that involves incorporating security into the continuous integration, continuous delivery, and continuous deployment pipeline. The product's security is established by including security testing and active security audits into the agile development and DevOps workflows.

2.2 Roles in DevOps

As one can notice from the name of the philosophy, the main roles in DevOps are related either to development or operations. Each role in the DevOps pipeline plays a critical role in ensuring the delivery of high-quality software quickly and reliably. Naturally, each team can be different due to the differences in a final product or team budget. Usually, the DevOps team consists of the following members: DevOps Engineer, Developer, Operations Engineer, Quality Assurance Engineer, Release Manager, Security Engineer, and Product Owner.

A **DevOps Engineer** is responsible for designing, implementing, and maintaining the DevOps pipeline, such as establishing Continuous Integration and Deployment (CI/CD), transferring data to the cloud, implementing the best tools and practices for automation, overseeing all technical processes, and offering IT support that is available on-call. The specific technical skills needed for a DevOps engineer may differ based on the team's structure, technology, and tools used. However, excellent communication and collaboration abilities are crucial. (Churylov, 2023)

Developers are responsible for writing the code that makes up the software application. In DevOps, developers work closely with other teams to integrate their code into the DevOps pipeline and ensure that it is tested and deployed correctly. Typically, the team leader in the development team is the most experienced engineer. They are responsible for assigning tasks to other team members, inspecting the code they generate, and executing the most intricate technical functionalities. (Churylov, 2023)

An **Operations Engineer** is responsible for managing the infrastructure and ensuring that it is available and performing well. They collaborate with other team members to develop and maintain deployment pipelines, automate processes, and monitor systems to detect and resolve issues. Additionally, they are involved in capacity planning, performance optimization, and disaster recovery planning. Operations Engineers play a critical role in maintaining high availability, reliability,

and scalability of the software and infrastructure in the production environment. While there is some overlap between the roles of Operations Engineer and DevOps Engineer, the main difference lies in the environment: Operations engineers focus on the production environment, while DevOps Engineers are responsible for managing the entire software development lifecycle.

The main objective of a **Quality Assurance (QA) Engineer** in DevOps is to guarantee that software systems and products conform to the predetermined quality standards (Churylov, 2023). To accomplish this goal, the QA engineer collaborates with developers, testers, and other stakeholders to develop and implement test strategies and automate test cases as much as feasible. The QA engineer is also accountable for detecting and reporting issues and tracking their resolution. Moreover, they strive to enhance the overall quality and efficiency of the development process by identifying areas for improvement and implementing remedies. The QA engineer's job is crucial in ensuring that the software being developed and deployed satisfies the end-users' requirements and is of excellent quality.

The duties of a **Release Manager** in DevOps are comparable to those of a project manager in a conventional IT team (Churylov, 2023). Their responsibilities include project planning, overseeing daily operations using Agile methods, and minimizing potential issues. However, unlike conventional managers, release managers in DevOps are also involved in technical aspects such as product development, integration, testing, and deployment.

For smaller projects, the duties of a **Security Engineer** and a DevOps Engineer can be merged into one role (Churylov, 2023). However, for larger teams, there may be specialized Security & Compliance Engineers who work closely with developers to ensure that their code and infrastructure are created with security in mind. These individuals are usually involved throughout the entire lifecycle of the product to guarantee compliance with security standards and regulations.

A **Product Owner** serves as a bridge between the DevOps team and customers. The specific requirements for a PO may vary depending on the project, ranging from an individual knowledgeable about the business and its customers, to a representative from an outsourcing firm. Primarily, a PO is responsible for communicating with stakeholders to establish a cohesive vision for the product. This includes developing a comprehensive product roadmap, determining which features take priority on the team's backlog, and assessing the team's progress. Furthermore, they collaborate with customers to obtain a better understanding of user requirements and improve the value delivered in each new product release. (Churylov, 2023)

2.3 DevOps frameworks

Frameworks in DevOps are sets of guidelines, practices, and tools that provide a structure for implementing DevOps principles and practices in software development and delivery. These frameworks help organizations to adopt a systematic approach to implementing DevOps and enable teams to work together more efficiently and effectively.

The need for DevOps frameworks arises because DevOps is a complex and constantly evolving discipline that involves various areas such as development, operations, security, and quality assurance. Implementing DevOps without a structured approach can lead to confusion, miscommunication, and inefficiencies. DevOps frameworks provide organizations with a clear roadmap for implementing DevOps practices and principles in a consistent and repeatable manner, making it easier to achieve the desired outcomes such as faster time-to-market, improved quality, and better customer satisfaction.

CALMS is a framework designed to evaluate a company's capability to adopt DevOps practices, and it is also used to measure progress during a DevOps transformation. The term was coined by Jez Humble, who is a co-author of the book "The DevOps Handbook," and the acronym represents Culture, Automation, Lean, Measurement, and Sharing**.** (Buchanan, 2023)

In DevOps, there are five key elements that make up the CALMS framework. The first is **Culture**, which pertains to the shared values, attitudes, and behaviors that shape how individuals and teams work together. DevOps culture emphasizes collaboration, communication, and a customer-focused approach. Another key element is **Automation**, which involves using technology to automate processes and increase efficiency in software development, testing, deployment, and operations. A **Lean** approach is also essential, where principles and practices are adopted to reduce waste, continuously improve processes, and deliver value to customers quickly. **Measurement** is another element, which uses metrics and data to evaluate the effectiveness of DevOps practices and identify areas for improvement. Lastly, **Sharing** is emphasized by promoting knowledge sharing, collaboration, and transparency to improve communication, break down silos, and align objectives. (Buchanan, 2023)

Other framework worth mentioning is **Team Topologies**, which came from the book written by Matthew Skelton and Manuel Pais. In their book, the authors describe essential team types and the idea of "change of flow." When implementing DevOps, it is not effective to merely add the label "DevOps Team" or "DevOps Engineer" to a job title without proper implementation of the right processes and tools. Instead, Team Topologies can help companies and organizations gain insight into how their practices and tools align with their overall goals. (Buchanan, 2023)

The book Team Topologies provides useful guidance on how organizations can integrate DevOps into their operations, including which team types are most effective. Before starting a DevOps transformation, it is important to identify the current organizational structure. However, many companies have multiple team types, and sometimes a single team may take on multiple roles and responsibilities, making it hard for leaders to have a complete understanding of the organization. Some important questions that leaders should be able to answer about their organizational structure include whether the current teams are suitable for the job, whether any areas of the organization lack the necessary

capabilities to meet their goals, and whether teams are receiving the right level of autonomy and support from other teams.

By using Team Topologies as a framework, leaders in development and operations can gain a clearer understanding of whether the right teams are in place. Atlassian suggests using four fundamental Team Topologies that can be easily understood by both upper management and team members. However, these team types may vary depending on the size and maturity of the organization, and in many cases, a combination of team types or a team transforming into another may be the most effective approach. (Buchanan, 2023)

Stream-aligned teams are specialized in a single, impactful stream of work, which can be a specific product, service, feature, user journey, or user persona. These teams are empowered to build and deliver value to customers or users quickly, safely, and independently, without the need for other teams to perform parts of the work. Due to the nature of their work, stream-aligned teams are agile and work closely with customers, incorporating their feedback into development cycles while maintaining software in production. While this type of team is common in many software companies, other organizations may be more accustomed to functional team structures (e.g., separate teams for engineering, design, QA) rather than delivery stream. Since stream-aligned teams are prevalent in organizations, other teams (i.e., complicated subsystem, enabling, and platform) are defined relative to them. To continuously improve the speed of delivery and quality of their products and services, stream-aligned teams should regularly collaborate with these supporting teams. (Buchanan, 2023)

Platform teams play a vital role in empowering stream-aligned teams to deliver their work with significant autonomy. The stream-aligned team is responsible for building, running, and fixing an application in production while the platform team provides internal services that the stream-aligned team can utilize. The platform team builds capabilities that are used by multiple stream-aligned teams with minimal overhead. By optimizing the product, the platform team reduces the resources and cognitive loads of the stream-aligned team. Furthermore, end-

users benefit from the platform team's work as they can create a seamless experience that spans across different user experiences or products. (Buchanan, 2023)

A **complicated-subsystem team** is responsible for building and maintaining a part of the system that depends on specific skills and knowledge. Most team members must be specialists in a particular area of knowledge to understand and make changes to the subsystem. The goal of this team is to reduce the load of stream-aligned teams who work on systems that include or use the subsystem. With the complicated-subsystem team's expertise and capabilities, stream-aligned teams don't have to build capabilities in areas too complicated for their daily work. Team members from this team may have specialized knowledge in certain microservices (i.e., a billing service), algorithms, or even artificial intelligence. A common pitfall is to embed specialists in every stream-aligned team who uses the subsystem. While this may seem efficient, it's ultimately not cost-effective and out of scope for a stream-aligned team. (Buchanan, 2023)

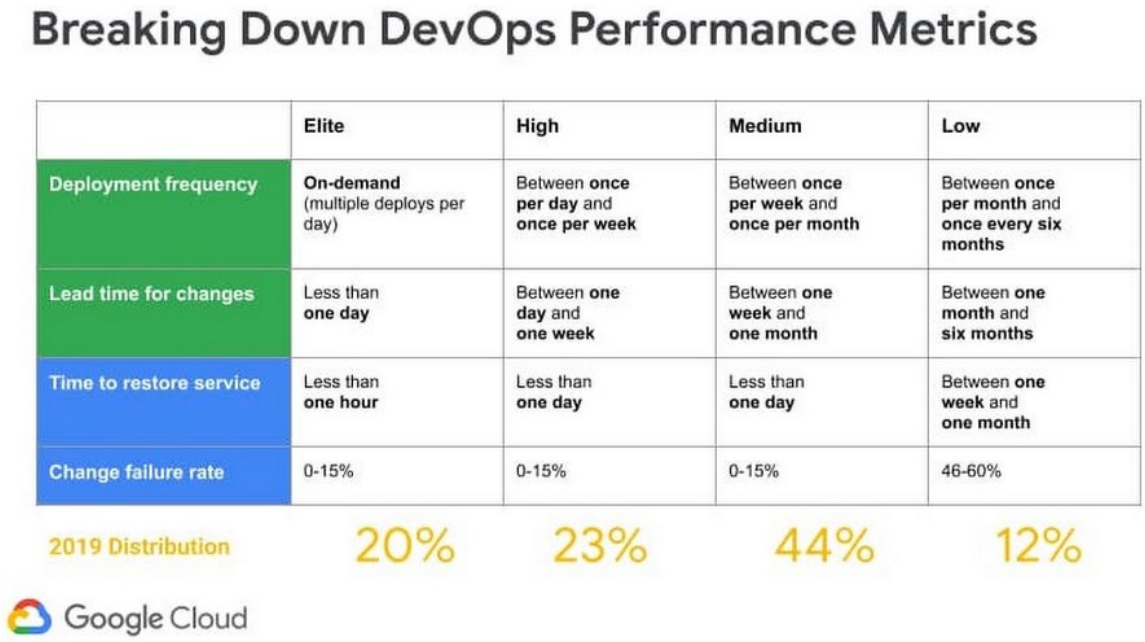
Stream-aligned teams face a constant challenge to deliver and adapt quickly, leaving little time for learning and exploring new skills. To overcome this challenge, organizations can rely on **enabling teams** composed of specialists in a particular technical or product domain. Enabling teams focus on research and experimentation to provide informed recommendations on tooling, frameworks, and ecosystem choices that impact the technology stack. This approach allows stream-aligned teams to build and develop their capabilities without sacrificing their primary goals. The primary goal of enabling teams is to increase the autonomy of stream-aligned teams by focusing on problem-solving rather than providing solutions. If the enabling team is successful, the stream-aligned team they support should be able to function independently in a few weeks or less, and the enabling team should not become a permanent dependency. (Buchanan, 2023)

Teams in DevOps use special metrics to measure success. Metrics in DevOps, also known as **DORA** (DevOps Research and Assessment) **metrics,** are specific

data points that provide insights into the efficiency of a software development pipeline and facilitate the detection and resolution of bottlenecks. They enable monitoring of technical capabilities and team processes. (Hall, 2023)

Fundamentally, DevOps emphasizes the collaboration between development and operations teams, erasing the boundaries between them. By using metrics, DevOps teams can measure and evaluate collaborative workflows, and keep track of progress towards achieving overarching goals such as enhanced quality, faster release cycles, and better application performance. Generally, DevOps performance is measured through the following four metrics (Table 1).

Table 1. Breaking Down DevOps Performance Metrics (Kaufman, 2020)



Lead time for changes (LT) is a crucial metric for measuring DevOps performance, and it differs from cycle time. This metric measures the time between when a code change is committed to the trunk branch and when it is ready for deployment, meaning it has passed all necessary pre-release tests. Lead times can be measured in hours for high-performing teams, while medium and low-performing teams often measure lead times in days, weeks, or even months. To improve lead time, practices such as test automation, trunk-based development, and working in small batches are key. These practices enable developers to receive fast feedback on the quality of their code commits, allowing

them to identify and remediate any defects quickly. On the other hand, if developers work on large changes that exist on separate branches and rely on manual testing for quality control, long lead times are almost guaranteed (Hall, 2023).

The second important DevOps metric to measure is the **change failure rate (CFR)**, which refers to the percentage of code changes that need to be addressed through hot fixes or other corrective actions after they have been deployed in production. This metric does not take into account failures detected during testing and fixed prior to deployment. Change failure rates in high-performing teams typically range from 0 to 15 percent. By implementing practices such as test automation, trunk-based development, and working in small batches, teams can reduce their change failure rates. These practices make it easier to identify and fix defects. Tracking and reporting change failure rates is also essential to ensure that new code releases meet security requirements in addition to identifying and fixing bugs (Hall, 2023).

The measurement of how often new code is released into production is a crucial aspect in assessing the success of DevOps, known as **deployment frequency (DF)**. Some DevOps professionals distinguish between "delivery" for code changes released into pre-production staging environments, and "deployment" for those changes that are released into production. Teams that perform at a high level can deploy changes at any time they want, and may even deploy changes multiple times per day. Conversely, lower-performing teams may only deploy changes on a weekly or monthly basis. To enable on-demand deployment, teams need to establish an automated deployment pipeline that integrates automated testing and feedback processes as discussed earlier. By minimizing the need for human intervention, teams can significantly reduce the time and effort required to deploy code changes (Hall, 2023).

The fourth essential metric that every DevOps team should track is the **mean time to recovery (MTTR)** (Hall, 2023). Another possible name for this metric is **time to restore service** (Kaufman, 2020). It is the duration between the time a

service interruption occurs and the time it takes to recover from it. It is an important metric to measure, regardless of whether the failure is due to a recent deployment or a system failure. High-performing teams excel in recovering from system failures quickly, usually in under an hour. Conversely, lower-performing teams may take up to a week to bounce back from a failure. Being able to recover promptly from a failure hinge on the team's ability to rapidly detect when a failure occurs and quickly deploy a fix or rollback any changes that triggered the failure. This is typically achieved by continuously monitoring the system's health and notifying operations staff in case of a failure. The operations staff must possess the appropriate processes, tools, and authorizations to resolve incidents. MTTR, as opposed to mean time between failures (MTBF), is the key focus of modern DevOps practices. MTTR acknowledges the increased complexity of contemporary applications and the likelihood of failure, promoting continuous learning and improvement. Instead of waiting for a "perfect" deployment that avoids any failure, teams continuously deploy. Instead of assigning blame for disrupting a "perfect" MTBF record, MTTR encourages blameless retrospectives to assist teams in enhancing their upstream processes and tooling. (Hall, 2023)

2.4 DevOps tools

DevOps approach uses various tools, depending on the project lifecycle phase. There is no single opinion regarding the overall number of phases. In Figure 1 below, eight phases are mentioned. However, some phases can be consolidated into a more general phase, such as Code and Build into Development phase. In this thesis, the DevOps lifecycle phases are Planning, Development, Testing, Deployment, and Monitoring.

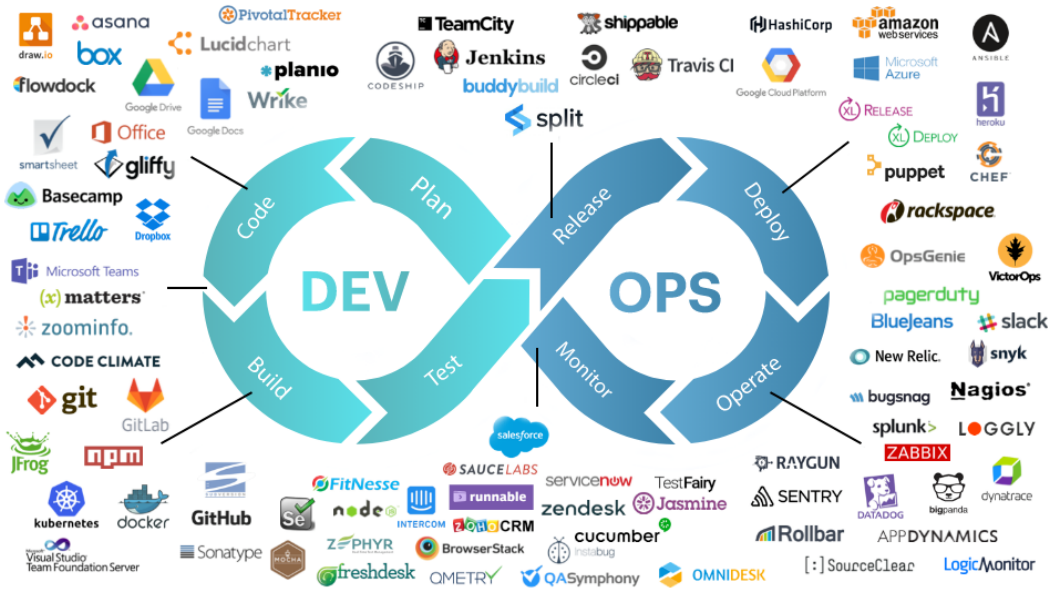


Figure 1. Example of DevOps lifecycle and suggested tools for various phases (Openxcell, 2023)

The **planning** phase is a crucial aspect of the DevOps lifecycle, as it serves as the foundation for the entire software development and delivery process. During this phase, the DevOps team collaborates to establish the project's scope, goals, objectives, resources, and timelines. Defining the software application requirements is one of the main aims of the planning phase, which entails comprehending the users' and stakeholders' needs. Afterwards, the DevOps team sets clear objectives and goals to ensure everyone is working towards the same end goal. Resource and timeline planning are also critical components of the planning phase, identifying the team members, tools, and technologies needed to build, test, deploy, and monitor the software application. Many DevOps teams utilize agile methodologies, such as Scrum or Kanban, to break the project down into smaller, more manageable tasks or user stories that can be completed in short sprints. Collaboration and communication are emphasized in the planning phase between the DevOps team and other stakeholders, such as business analysts, product owners, and customers, to ensure everyone is aligned on the project's goals, objectives, resources, and timelines (Pennington, 2019).

The planning phase utilizes various tools, including Jira, Trello, Confluence, Asana, ReqView, and ReQtest for agile project management and requirements

management. Additionally, communication and collaboration tools like Slack, Microsoft Teams, and Zoom are used. (Atlassian, 2023)

The **development** phase is a critical stage of the DevOps lifecycle where software applications are constructed, tested, and readied for deployment. Here, developers write code to realize the requirements and objectives determined during the planning phase. In this phase, the continuous integration and continuous delivery (CI/CD) pipelines play an essential role. These pipelines automate the building, testing, and deployment processes, empowering developers to build, test, and deploy code changes quickly and reliably. To streamline code changes, developers employ various tools and technologies such as Git for version control, Gradle or Maven for build automation, and JUnit or Selenium for testing (Mohanan, 2022). Collaboration and communication between developers, testers, and other members of the DevOps team are vital in this phase to ensure everyone is aware of the progress and address any issues promptly.

The **testing** phase is the next phase of DevOps lifecycle, where the software application is thoroughly tested to ensure it meets the requirements and operates correctly. During this phase, various testing activities are conducted, including functional, performance, security, and user acceptance testing. The primary objective of the testing phase is to identify defects and issues in the application and fix them before deployment. Automated testing is a critical component of the testing phase, which allows teams to test the application quickly and accurately at different stages of development. Test automation tools like Selenium, Appium, and JMeter can perform various types of tests automatically, including unit, integration, and acceptance testing, making the process faster and more efficient. The testing phase also involves using metrics to measure the effectiveness of the testing process (Mohanan, 2022). Metrics like defect density, code coverage, and test coverage can evaluate the quality of the application and the testing process. Testers provide feedback on the application's performance, and developers address any issues that arise, ensuring the application meets the requirements and functions as expected.

The **deployment** phase is a crucial step in the DevOps lifecycle, as it involves releasing the software application to end-users. In this phase, code changes are deployed to the production environment, and the application's functionality and stability are ensured. To reduce the risk of errors or downtime, continuous delivery and deployment (CD) pipelines are employed to automate the release process. Various tools and technologies such as Ansible, Chef, Docker, Kubernetes, AWS, and Azure are used to manage and deploy code changes efficiently. Smoke testing, monitoring, and rollbacks are performed to ensure a smooth deployment process (Mohanan, 2022). Monitoring helps in tracking the application's performance in production, while rollbacks involve reverting to a previous version of the application if any issues arise. Effective collaboration and communication are essential in this phase to keep all stakeholders informed of the deployment status and any issues that arise.

The final phase of the DevOps lifecycle is the **monitoring** phase, where teams monitor the application in production to ensure it meets service level objectives (SLOs) and performs as expected. Monitoring is essential to identify issues or errors that may impact end-users and ensure the application is available and responsive. Various tools and technologies are used during this phase, such as log analysis tools, monitoring tools like Nagios or Zabbix, and application performance monitoring (APM) tools like New Relic or Datadog. These tools provide real-time monitoring and alerts, enabling teams to quickly identify and resolve issues (Mohanan, 2022). The collected data is also analyzed to identify trends and patterns, which can be used to improve the application's performance, optimize it, and refine the SLOs. Effective monitoring is crucial for meeting end-users' needs and making informed decisions about future development and updates. Collaboration and communication among stakeholders are also necessary to resolve any issues or trends in the application's performance.

To successfully develop a DevOps product, the selection of an appropriate **toolchain** is crucial. A DevOps toolchain refers to a group of integrated tools, often from various vendors, that operate collectively to design, build, test,

manage, measure, and operate software and systems. It facilitates collaboration between development and operations teams throughout the product lifecycle and addresses key DevOps concepts such as continuous integration, continuous delivery, automation, and collaboration. To determine the suitable DevOps toolchain, it is essential to comprehend the fundamental DevOps practices and how tools can support these practices. After that, it is vital to establish a shared tool strategy that enables teams to collaborate during development, testing, and deployment. When implementing DevOps, organizations typically face two options: an all-in-one DevOps toolchain or a customized DevOps toolchain. As the toolchain selection shapes the DevOps processes of a team, choosing the correct configuration is crucial. (Krohn, 2023)

An **all-in-one DevOps toolchain** solution that provides all the necessary tools in a single package can be a valuable option for companies that are new to the DevOps process or those that need to start a project quickly. However, one drawback of an all-in-one toolchain is that established teams may already have a preferred set of tools that do not integrate with a comprehensive solution. Additionally, such a toolchain may suffer from the "jack of all trades, master of none" syndrome, where a single tool cannot keep up with rapidly changing markets. Finally, many companies need to integrate legacy tools into their DevOps toolchain, which an all-in-one solution may limit. (Krohn, 2023)

A different approach to DevOps toolchains is to opt for **customized DevOps toolchain**, whereby teams can choose different tools that meet their specific needs. This allows teams to integrate their existing preferred tools into the wider DevOps toolchain. For instance, Jira can be used for planning and workflow tracking, Kubernetes can be used to provide individual development environments, Github can be used for collaborative coding, Jenkins can be used for continuous integration, and so on. Workflows can be customized either by teams or projects within organizations. For this type of toolchain to be effective, integration is crucial. If the tools being used do not integrate well with each other, team members may waste time switching between different screens and logging in to multiple accounts. This can also make it challenging to share information

between tools, leading to an unpleasant experience for developers or anyone trying to understand what's going on. When responding to an incident, there may not be sufficient time to go through manuals and look up information about unfamiliar tools. (Krohn, 2023)

2.5 Relationship to other approaches

There are other methodologies which share similarities to DevOps approach in software development. However, there are also significant differences.

Agile methodology, together with its frameworks, such as Scrum and Kanban, focuses on iterative and incremental development, emphasizing collaboration among development teams and stakeholders to continuously deliver value to customers. The methodology emphasizes flexibility and responsiveness to change, allowing necessary adjustments during development. The primary objective of Agile is to deliver working software promptly, ensuring it meets customer requirements (Atlassian, 2023). Both Agile and DevOps prioritize collaboration among teams, prioritize communication and transparency, and emphasize the need for cross-functional teams to work together. Additionally, they share the importance of delivering functional software that meets customer needs.

Despite some similarities, Agile and DevOps have notable differences. Agile methodology aims to deliver customer value through iterative development, while DevOps concentrates on collaboration and communication among teams to ensure quick deployment and continuous delivery. The primary focus of Agile is on the development process, whereas DevOps emphasizes the operational side of software delivery. Moreover, Agile teams are typically cross-functional, with developers, testers, and business stakeholders working together to deliver software. In contrast, DevOps teams typically include developers and IT operations personnel, who work together to manage the software delivery and deployment process. Agile methodology focuses on continuous integration (CI) and continuous delivery (CD), which involve automatically building, testing, and deploying code changes to a test environment. Conversely, DevOps takes a

broader view of software delivery, encompassing not just CI and CD, but also the entire software delivery pipeline, from development to production.

Although **Continuous Delivery** and DevOps are sometimes used interchangeably, they are actually two distinct concepts (Hammond, 2011). The main similarity of these two approaches is the aim to improve the software development process. While DevOps focuses on a cultural shift that emphasizes collaboration among different teams involved in software delivery, automation of software delivery processes, and improvement of the entire software development lifecycle (Humble & Farley, 2011), Continuous Delivery mainly focuses on automating the delivery aspect of the software development process, aiming to improve the speed and frequency of software releases by integrating various processes (Swartout, 2012). Therefore, DevOps has a broader scope compared to Continuous Delivery. Continuous Delivery can be seen as a prerequisite for DevOps, and it can be considered as the starting point towards achieving DevOps. Achieving DevOps requires more significant cultural change and organizational transformation than CD.

Another methodology is **Waterfall**, which is a sequential approach to software development, with each phase (such as design, implementation, testing, and maintenance) being completed before moving on to the next one. Waterfall is more structured and has a higher degree of documentation than Agile or DevOps. While Waterfall can be effective for large, complex projects with stable requirements, it is less flexible and can be slow to adapt to changes. Every change is difficult and expensive to implement (Wikipedia, 2023). DevOps, on the other hand, embraces change and seeks to incorporate feedback and new requirements into the development process as quickly as possible. Waterfall emphasizes quality assurance during the testing phase, while DevOps seeks to build quality into the development process from the beginning. By automating testing and incorporating testing earlier in the development process, DevOps aims to ensure that the software is of high quality and meets customer needs. Overall, DevOps approach is totally different from Waterfall methodology.

DevSecOps is a methodology that aims to integrate security practices into the DevOps process, making security an integral part of the software development lifecycle right from the start. In the past, security was often treated as a separate process that came after development and deployment, which could result in inefficiencies and security breaches. By incorporating security into the development process, DevSecOps enables developers to identify and address potential security risks at every stage of development. Figure 2 demonstrates security tasks in a DevSecOps pipeline, breaking down the process into pre-production and production stages. (Zettler, 2023)

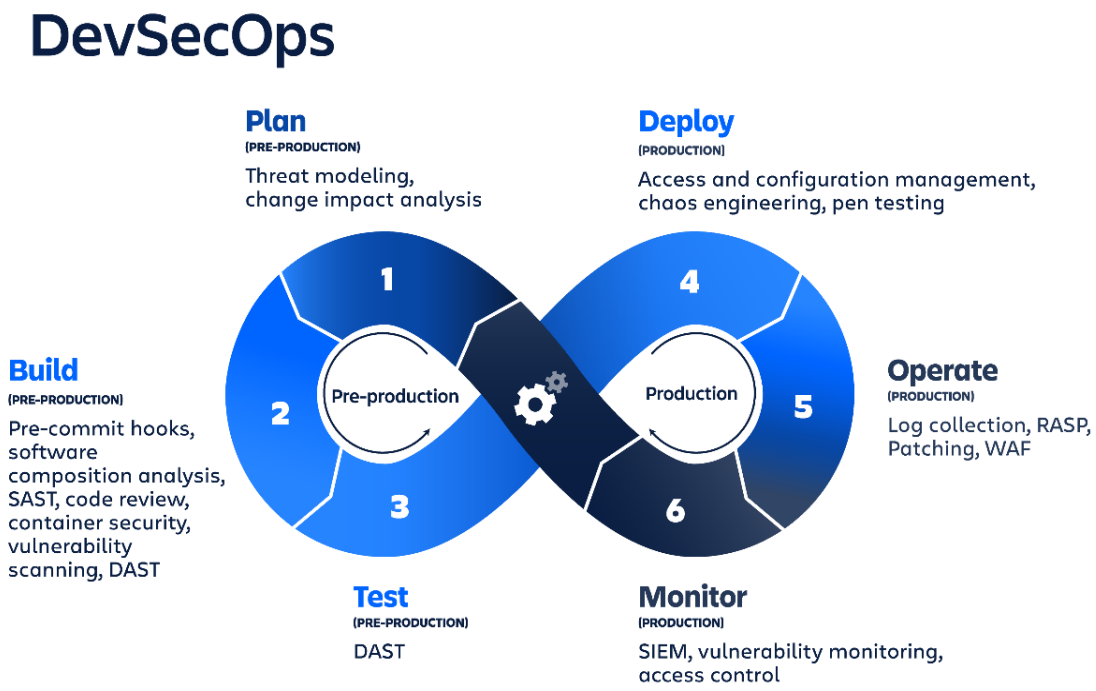


Figure 2. Pre-production and Production security tasks in DevSecOps (Zettler, 2023)

DevSecOps promotes a culture of shared responsibility among development, security, and operations teams. Developers are encouraged to write secure code and perform regular security testing, while security teams provide guidance and feedback on best security practices. Operations teams ensure that the application is secure and available in the production environment. The benefits of DevSecOps include increased security, reduced risk, and faster delivery of secure software. By identifying and addressing security issues early in the development process, teams can reduce the risk of security breaches. Integrating security into the development process also ensures that security is not an

afterthought but is an integral part of the software development lifecycle, leading to improved collaboration among teams and faster delivery of secure software.

2.6 Atlassian systems

Atlassian is a company that creates and offers an assortment of software tools and products designed for software developers and teams to manage and work together on their projects. Among its widely used product suite are tools like Jira which provides issue tracking and project management capabilities, Confluence for team collaboration and documentation, Bitbucket which provides code hosting and version control, and many other products. The tools developed by Atlassian are popular in the software development industry and are renowned for their flexibility, scalability, and seamless integration with numerous third-party systems and tools.

The Open DevOps solution by Atlassian presents a comprehensive DevOps process by integrating Atlassian and third-party tools. Teams can make use of Atlassian products or their preferred products in the open toolchain, with Jira serving as the main component. Atlassian's ecosystem includes a wide range of integrations and add-ons, enabling teams to personalize their toolchain according to their requirements (Atlassian, 2023). Figure 3 demonstrates which Atlassian tools can be used for each phase in a DevOps lifecycle.

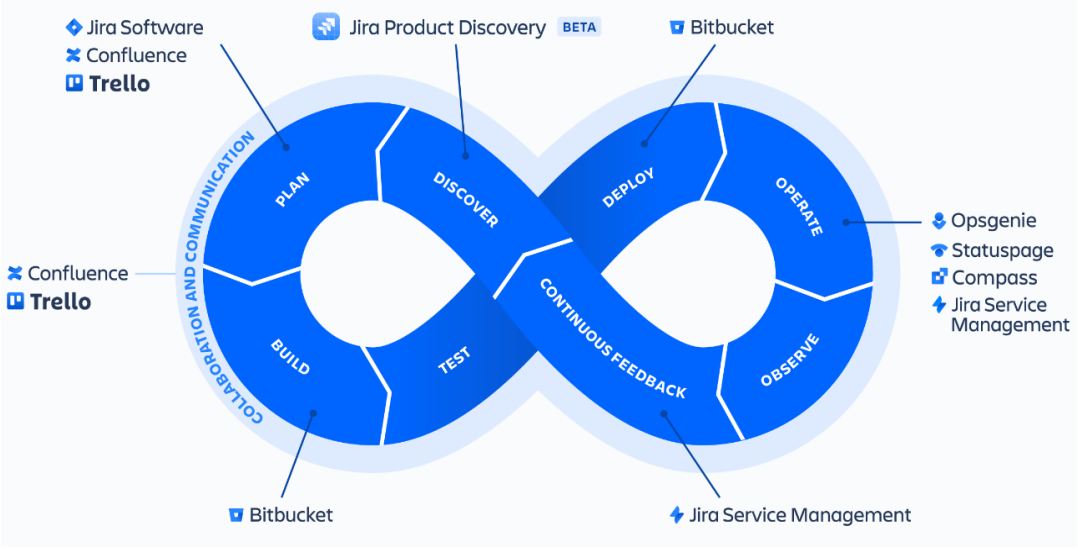


Figure 3. Atlassian products in a DevOps lifecycle

In the practical implementation part of the thesis, Atlassian products are utilized for implementing the DevOps approach in software development.

2.6.1 Jira

Atlassian has developed Jira, a popular software for issue tracking and project management, which offers two different types: Jira Software and Jira Service Management. Each of these products has distinct functionalities and serves different purposes.

Jira Software is a tool for software development that allows teams to manage their projects, track progress, and collaborate in real-time. It is equipped with features like backlog management, sprint planning, and user story tracking, supporting agile project management methodologies. Software development teams that utilize agile methodologies and require a powerful and adaptable platform to manage their workflow will find Jira Software useful. The key element in Jira is issue. Jira issues can take on various forms depending on how a team uses the platform. They can represent project tasks, helpdesk tickets, leave request forms, and more. In Jira Software, however, issues generally pertain to individual work items such as major features, user requirements, and software bugs.

Jira Service Management, on the other hand, is an IT service management solution aimed at helping IT teams provide efficient and effective services to customers. It offers features like incident management, problem management, change management, and service request management. The platform is designed to be flexible and scalable, allowing IT teams to track customer requests and incidents, automate workflows, and analyze service performance.

Both Jira Software and Jira Service Management are highly flexible and customizable tools, enabling users to adjust workflows, fields, and tags according to their organization's specific requirements. They are also highly scalable, accommodating teams of all sizes, from small startups to large enterprises.

Despite similarities, Jira Software and Jira Service Management have significant differences. While Jira Software is focused on software development, Jira Service Management concentrates on IT service management. Jira Software is a suitable solution for software development teams, while Jira Service Management is intended for IT teams that manage and deliver services to customers.

Jira Software serves as the foundation of Atlassian's Open DevOps, an integrated toolchain for DevOps teams. It is designed to work seamlessly with both first- and third-party tools throughout the DevOps lifecycle, including code and version control tools such as GitHub, GitLab, and Bitbucket, documentation and knowledge management tools like Confluence, and monitoring and operating tools such as Opsgenie. In addition, Jira Software can integrate with various tools across different categories to help DevOps teams enhance their software development practices and accelerate the delivery of high-quality software. (Atlassian, 2023)

Both products offer two hosting options: cloud or data center hosting. Atlassian's cloud products offer a comprehensive Software-as-a-Service (SaaS) solution that is readily available to all customers. With the cloud option, customers can rely on Atlassian to handle the heavy lifting by providing built-in security and compliance features, hassle-free setup, and financially-backed Service Level Agreements (SLAs) for uptime and performance. Moreover, cloud products continuously evolve, giving end-users access to the latest and greatest features and functionality. On the other hand, Atlassian's Data Center products, also known as "self-managed," provide customers with the flexibility to deploy on their preferred infrastructure. This option allows customers to manage data, security, and compliance, decide when to upgrade, and control uptime and performance management. (Atlassian, 2023)

In the practical part of the thesis, Jira Software plays the most vital role for implementing solutions of DevOps philosophy.

2.6.2 Confluence

Atlassian's Confluence is a web-based tool that supports collaboration and documentation (Atlassian, 2023). It enables teams to create, share, and organize content on a single platform, making it a valuable solution for knowledge management and teamwork.

The tool has an array of features such as macros, templates, and collaboration tools that aid teams in producing and organizing content effectively. Confluence is highly scalable and adaptable, accommodating organizations of all sizes, from startups to large enterprises, with customization options to match the needs of a specific team or business. It integrates seamlessly with other Atlassian tools, including Jira Software and Bitbucket, as part of a comprehensive ecosystem of tools that promote efficiency and collaboration, which are fundamental to the DevOps philosophy. Teams can work together and create various types of content, such as project plans, technical documentation, meeting notes, and product requirements. Confluence also has a centralized content repository, which allows easy search and reuse of information, making it a crucial component for DevOps. Its advanced search functionality further increases productivity by enabling users to find content rapidly.

Similar to Jira, Confluence can also be deployed on Cloud or Data Center, providing customers the freedom to choose the most suitable option for their business needs.

2.6.3 Others (Bitbucket, Opsgenie, Crucible, Fisheye)

There are other tools and products offered by Atlassian, nearly all of them include integration to each other. Aside from Jira and Confluence, the most important Atlassian tools for DevOps philosophy are Bitbucket, Opsgenie, Crucible and Fisheye.

Bitbucket is a web-based version control repository hosting service developed by Atlassian. It enables software development teams to collaborate on code, review

code changes, and manage their source code repository in a secure and scalable manner (Atlassian, 2023). As a part of the Atlassian toolchain, Bitbucket is designed to enhance the DevOps philosophy by enabling teams to continuously deliver high-quality software with speed and reliability. By providing features such as code review, branch management, and integration with other Atlassian tools like Jira and Bamboo, Bitbucket helps teams to improve collaboration, increase automation, and achieve greater transparency throughout the software development lifecycle.

Opsgenie is a modern incident management platform that helps DevOps teams quickly respond to and resolve incidents, enabling faster mean time to resolution (MTTR) and reducing downtime. With Opsgenie, teams can centralize alerts and notifications from multiple sources, such as monitoring and logging tools, and route them to the right team members, ensuring that the right people are notified at the right time. The platform also offers advanced features, such as on-call schedules, escalations, and incident analytics, allowing teams to optimize their incident management processes and continually improve their incident response (Atlassian, 2023). By providing a comprehensive incident management solution, Opsgenie can greatly assist in DevOps, helping teams achieve their goals of delivering high-quality software at a rapid pace while maintaining uptime and availability.

Crucible and Fisheye are tools that can help in DevOps by improving code quality and collaboration. Fisheye is a code search and navigation tool that enables developers to browse and search through code repositories, including Git, Subversion, and Mercurial (Atlassian, 2023). This allows teams to identify code changes quickly and easily, which is essential in a DevOps environment where speed is critical. Crucible is a code review tool that enables developers to review code changes and provide feedback in a collaborative manner (Atlassian, 2023). This ensures that code is reviewed thoroughly before it is released, which helps to improve code quality and reduce the risk of errors. By using Crucible and Fisheye together, DevOps teams can ensure that code is reviewed thoroughly and changes are implemented quickly and efficiently.

3 PRACTICAL PART

The practical part of the thesis examines real-life scenarios where the DevOps approach can aid users. It begins with an overview of the commissioner and my role in the work. Subsequently, detailed descriptions of automation cases and enhancements in users' work are provided, representing the central focus of this study. Finally, an overview is presented showcasing how these changes enabled teams to effectively manage their tasks and achieve successful delivery of high-quality products. All company-sensitive information is omitted in the study and screenshots blurred for privacy reasons.

3.1 Commissioner overview

Playtech is a software development company that specializes in providing online gaming and sports betting solutions. Founded in 1999, Playtech is one of the world's leading providers of online gaming software and has developed a wide range of products and services for online casinos, poker rooms, sportsbooks, and other gaming operators. Playtech's products include a variety of casino games such as slots, table games, and live dealer games. They also offer back-end management systems for gaming operators, payment processing solutions, and player management tools. Additionally, Playtech has developed a range of sports betting products including odds feed and trading tools.

My career at Playtech began as an intern in the summer of 2021. Officially, my job position is called Information Solutions Engineer. While it is not directly related to DevOps, occasionally, the practices of the DevOps philosophy appear at work, and its presence in my daily routine is noticeable. It is worth mentioning that the responsibilities of a DevOps Engineer are not fixed, and one does not necessarily have to be called so to implement the solutions and philosophy of the practice.

For the most part, my job responsibilities involve administering Jira and assisting users with their requests to help them become more productive and comfortable

with the working environment. For instance, I help optimize the working routine by setting up automation rules and integration to other services. By default, plain Jira offers basic capabilities, however, Atlassian offers some plugins for free, which greatly assist users in following the DevOps philosophy.

At work, I received some cases from team leaders who wanted to streamline the software development process. Additionally, I explored the capabilities and possible solutions of various plugins and applications, which inspired other ideas for the thesis.

3.2 Tools and plugins

As mentioned earlier in the theoretical part, Atlassian products demonstrate excellent compatibility and integration capabilities with a wide range of third-party solutions from various vendors. While there is a vast selection of plugins available, this study focuses on describing the potential of the most essential ones, based on my opinion.

The practical implementation of the thesis was done with the Jira Software Data Center 8.20. Jira plugins used for the study are Automation for Jira, ScriptRunner and Issue Matrix. Additionally, integration to Microsoft Teams is mentioned.

3.3 Test project setup

To begin with, we need to create a project where we can implement our solutions. Figure 4 illustrates possible templates a project can have. For the thesis, *Scrum software development* template is chosen. The option mainly affects a workflow that will be used in issues – however, it can be changed later. Scrum software development template also gives the opportunity to use boards, sprints and connects with build tools.

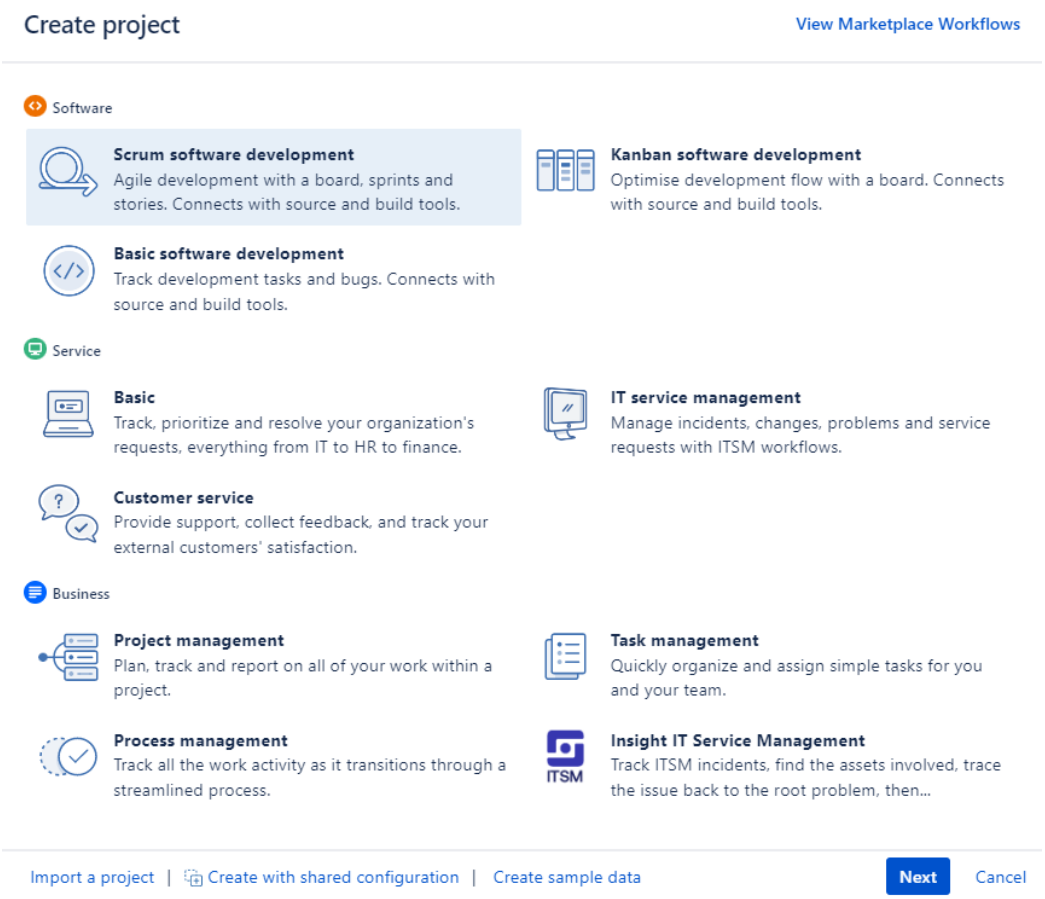


Figure 4. Project template window in Jira

Next, the window shows the issues types and the workflow given in the template (Figure 5). As mentioned earlier, this is just a template and the project settings can be changed any time by project administrators.

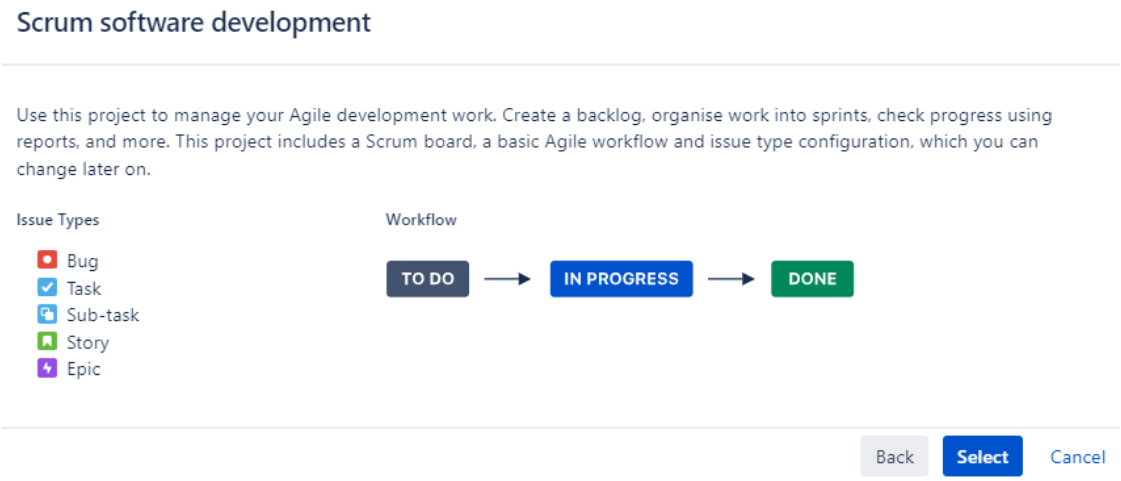


Figure 5. Scrum software development template workflow in Jira

Lastly, a name for the project and its key should be entered. Also, a project lead is specified – the main person responsible for the project. This stage can be seen in Figure 6.

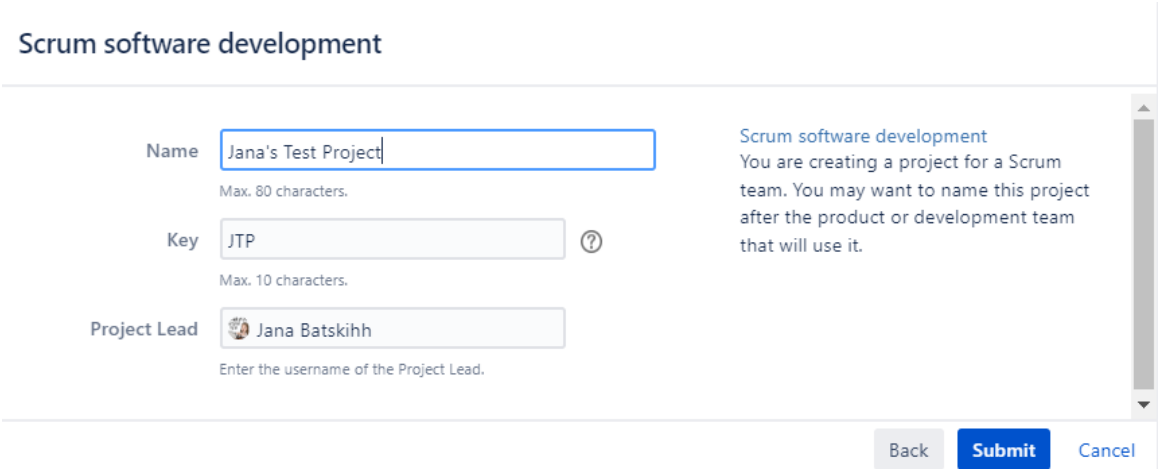


Figure 6. Project naming window in Jira

The test project is set up and ready to be modified.

3.3.1 Email and Microsoft Teams notifications about creating new issue

Notifications about the newly created Jira issue allow users to start working on the task as soon as possible without minimal delays, thus saving a considerable amount of time. This can be especially needed when there is a group of users where any of them can start working on the task (for example, someone from the developer team). The tool used in this automation is Automation for Jira – a free app offered by Atlassian.

In the *Project settings*, choose *Project Automation*, then *Create rule*. Make sure to create the rule in the *Project rules* section, otherwise the automation will apply to all projects in the instance, which may not be favorable and can cause performance issues if set up incorrectly. Figure 7 illustrates the location of automation settings in Jira project.

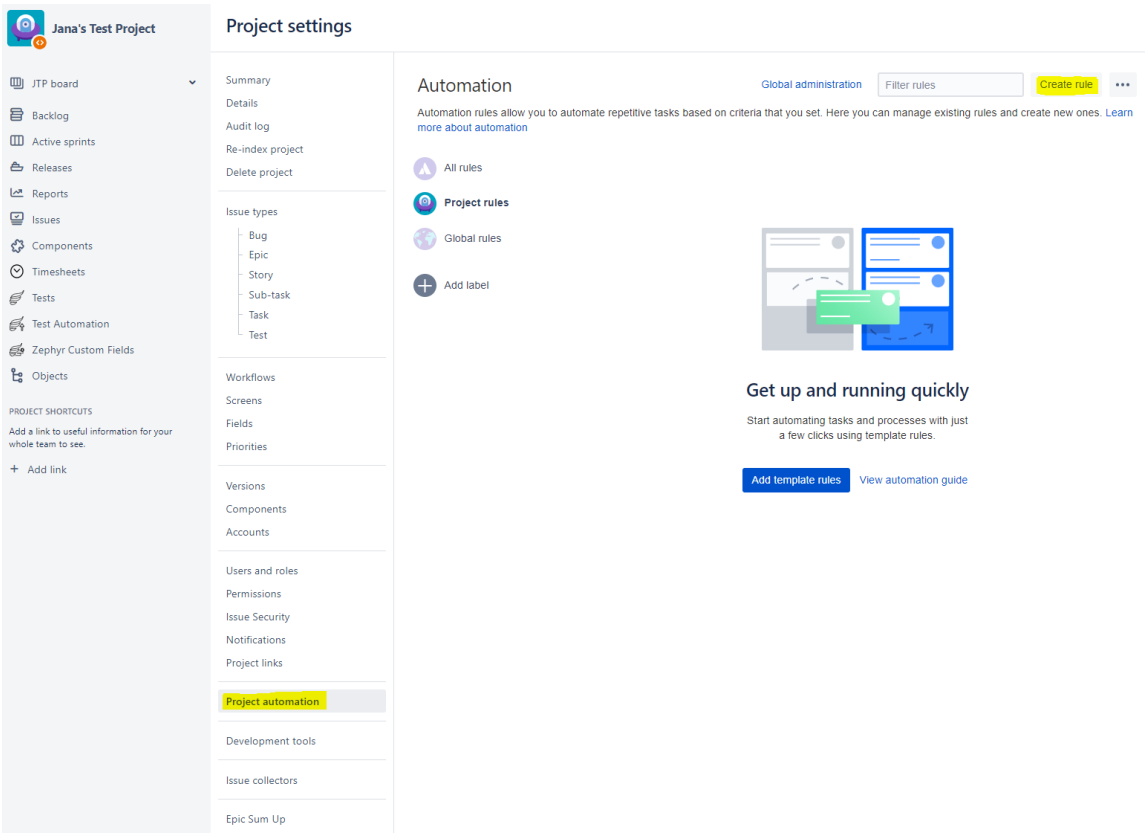


Figure 7. Project settings page with Project automation section selected

Triggers start the execution of a rule, so if we want to set up automation that notifies users about the creation of a ticket, then the trigger should be *Issue created* (as illustrated in Figure 8).

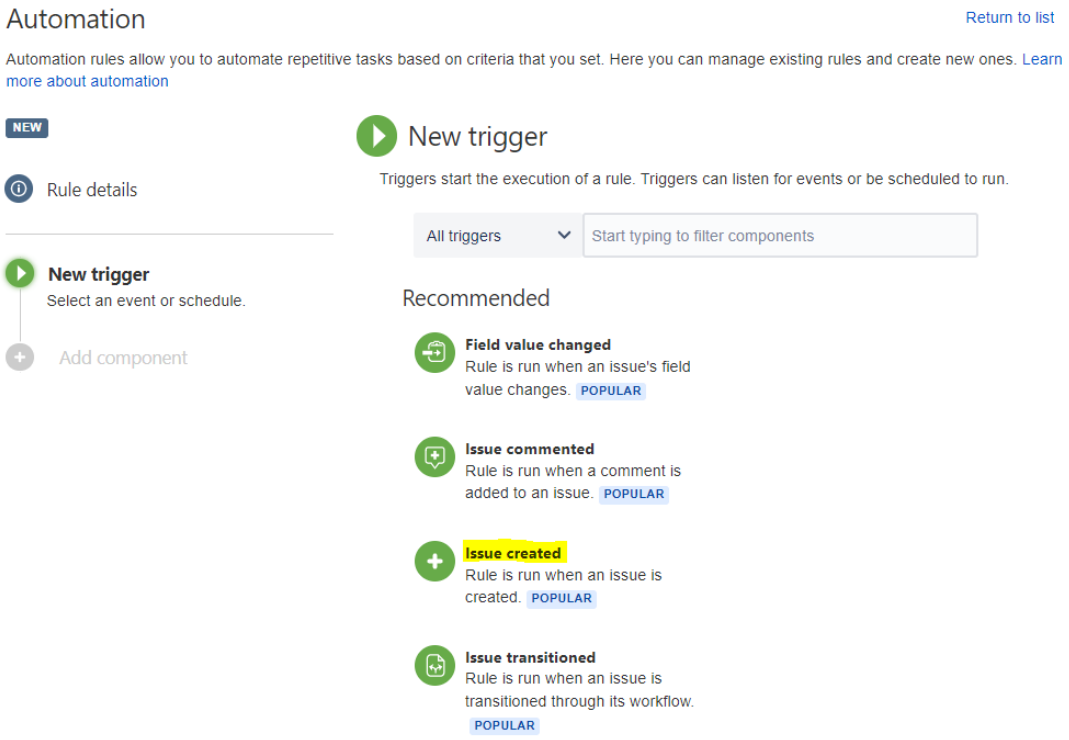


Figure 8. New automation rule creation, stage 1

Smart values can be used in composing the subject and content of the email message. Notice the smart values that indicate the issue key, issue summary and project name (Figure 9). The email can be set as an email distribution group too – for example, in case if a whole team/department has to know about issue creation.

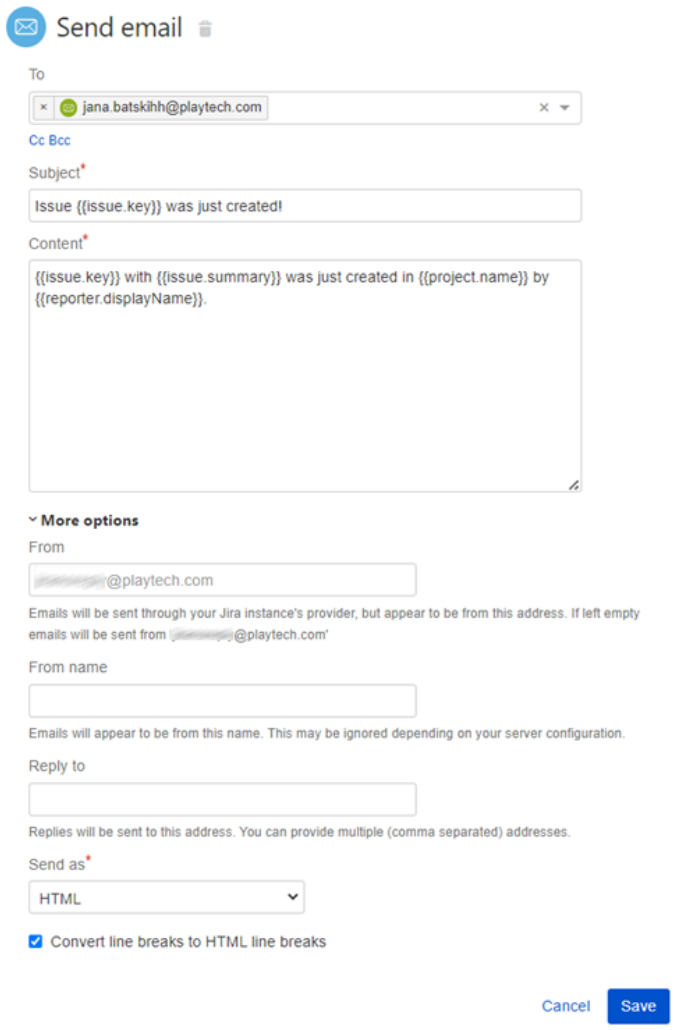


Figure 9. New automation rule creation, stage 2

Save and enable the automation rule. Test the solution by creating a new issue in the project. Final result is seen in Figure 10.

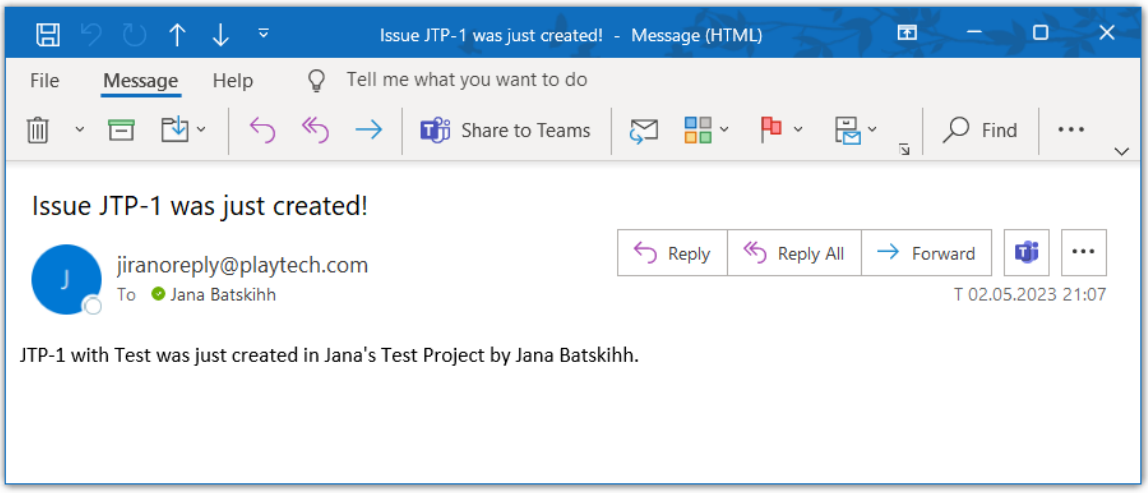


Figure 10. Automatic email sent to the user.

As an option, other needed information can be added for the email message. For example, description of an issue, its priority, or the link to it. Figure 11 demonstrates the solution.

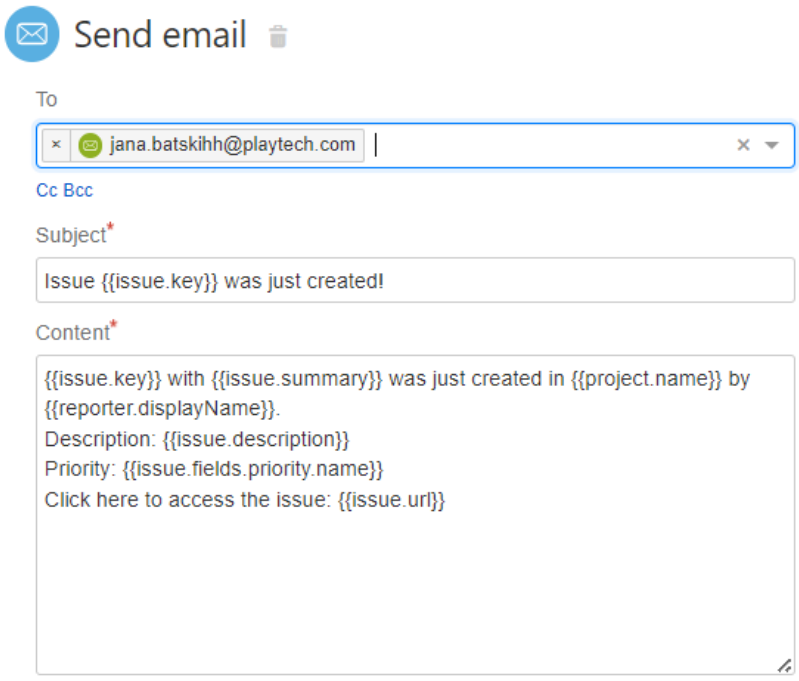


Figure 11. New automation rule creation, stage 2, second case

The outcome of the last automation can be viewed in the Figure 12.

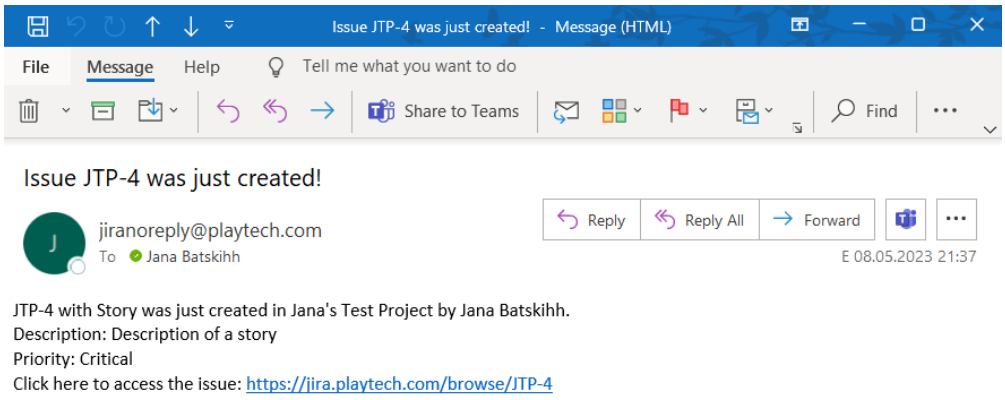


Figure 12. Automatic email sent to the user, second case

The other way to send notification is via Microsoft Teams. This approach uses native Jira functionality, however, requires installing Microsoft Teams for Jira Server application, which can be downloaded from Atlassian Marketplace for

free. Suppose that we want to create notification with the same logic – to update users that the issue is created.

In Jira, go to *System -> Advanced -> Webhooks -> + Create a WebHook*. Destination page is illustrated in Figure 13. Note that other webhooks in the column were blurred for privacy reasons.

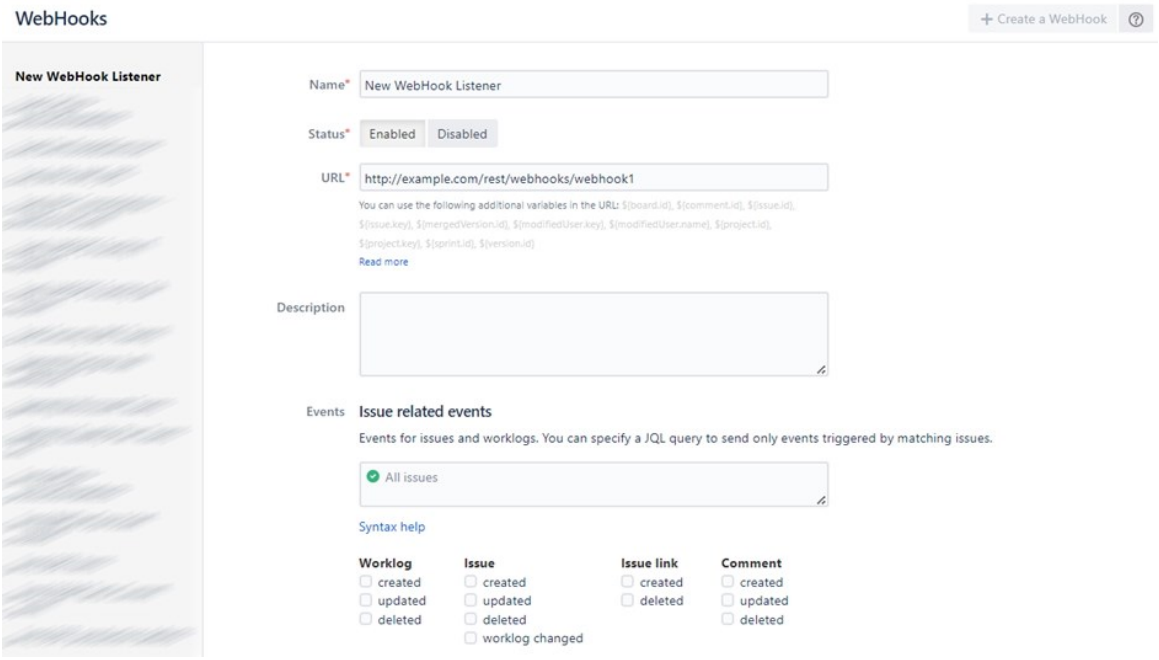


Figure 13. New webhook creation in Jira, stage 1

For the *Name*, choose one that can be easily identified. *Status* should be Enabled – so that the automation would work. *URL* will be updated later – this is the URL of the webhook that connects to Microsoft Teams. *Description* can be left empty. For the *Events* section, a JQL query should be set.

Jira Query Language, also known as JQL, is a highly robust and versatile tool for searching issues within Jira (Radigan, 2020). Write a JQL query that corresponds to the projects and its issue types you would like to receive notifications for. We will set the notifications for all issues created in *Jana's Test Project*. Before entering the query into the box, test that it works and returns values in *Issues -> Search for issues*. Write the query as detailed as possible. While we could leave only *project = "Jana's Test Project"*, it is better for performance reasons to specify the issue types used in this project too.

Final query, based on requirements above:
project = "Jana's Test Project" AND issuetype in (Bug, Task, Sub-task, Story, Epic)

You can also add additional requirements. For example, if you want to include only issues with priority of Critical or Emergency, write the following:
project = "Jana's Test Project" AND issuetype in (Bug, Task, Sub-task, Story, Epic) AND priority in (Critical, Emergency)

If you want to receive notifications for the critical and emergency issues, and on issues you are assigned to, no matter their priority is (note that this automation will work only if the assignee is added into the issue before creating/posting the issue; it will not work if the assignee field is updated after the issue is created):
project = "Jana's Test Project" AND issuetype in (Bug, Task, Sub-task, Story, Epic) AND (priority in (Critical, Emergency) OR assignee = currentUser())

As a next step, open Microsoft Teams. Create a new Team. Create a new channel – call it e.g., *Jira notifications*. Click on … to select *Connectors* and choose *Jira Server*. A configuration window should be seen as illustrated in Figure 14. Copy the URL, click *Save* and return to Jira webpage.

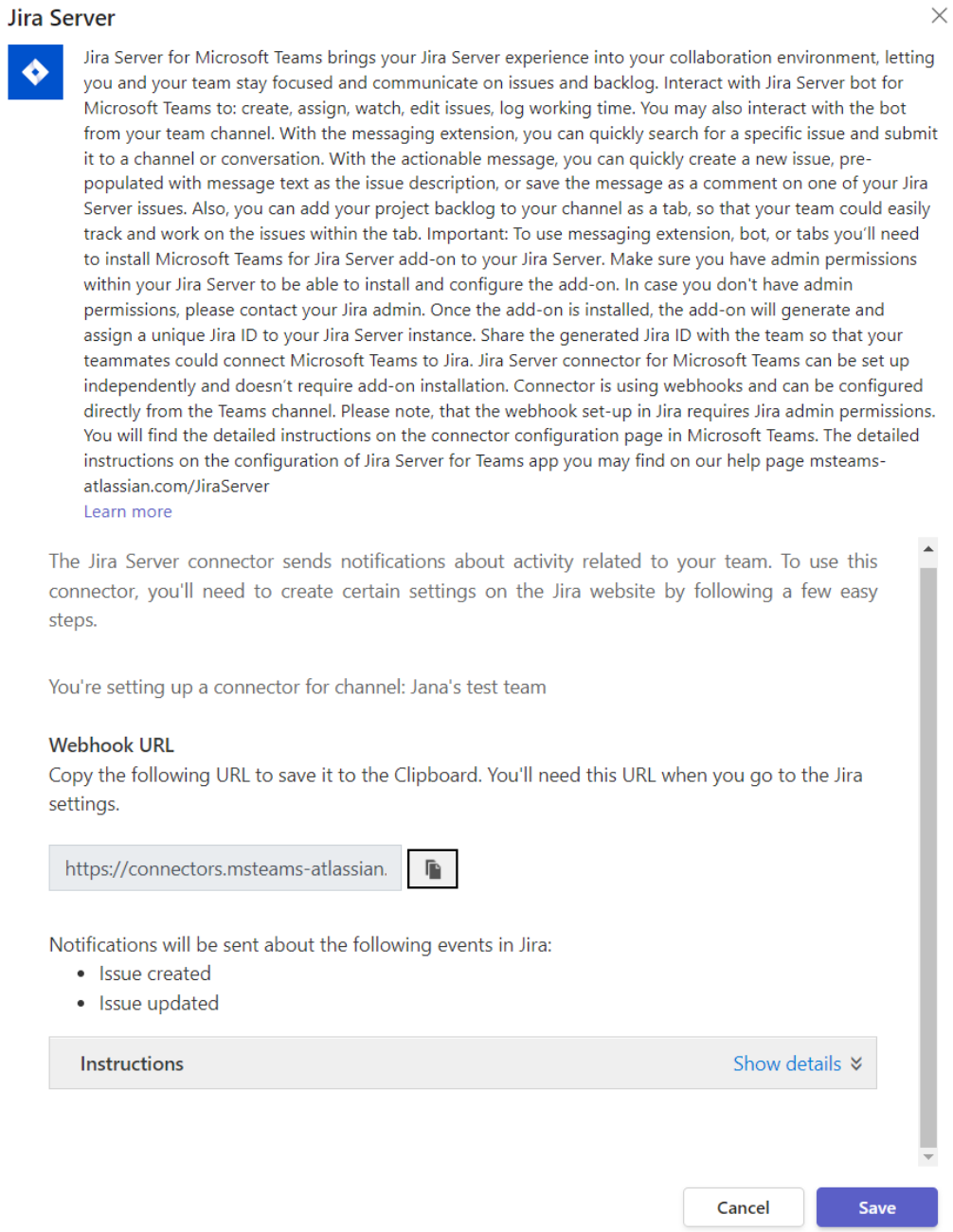


Figure 14. Jira Server Connector in Microsoft Teams

Figure 15 shows how the final options should look like. Note that we have also selected Issue created and Comment created boxes – this will let the webhook know when to fire the event.

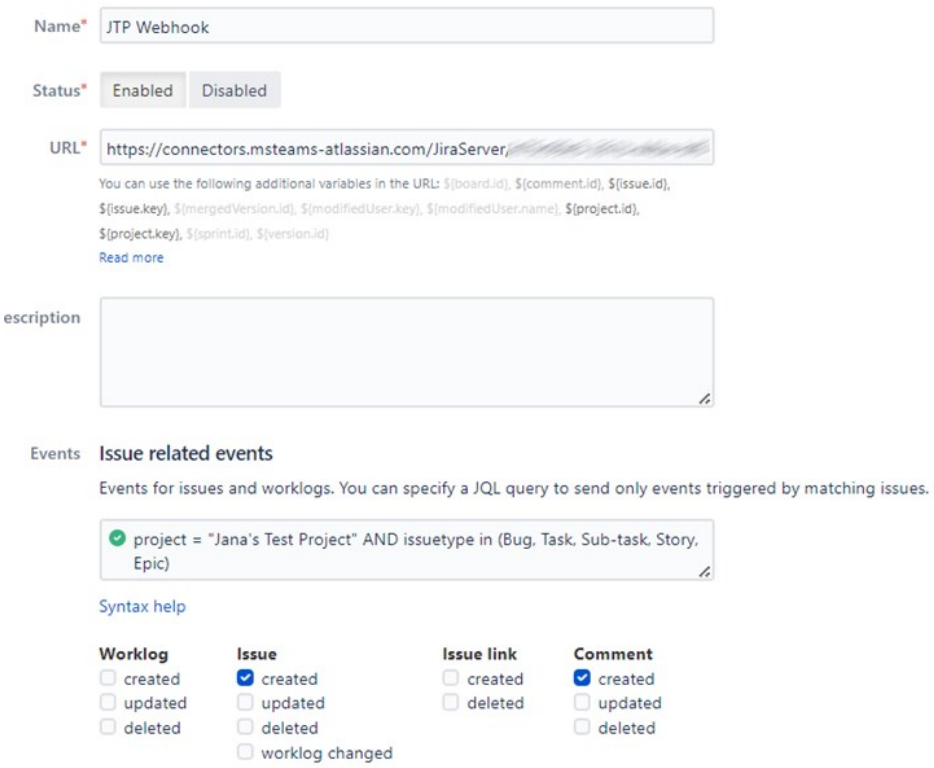


Figure 15. New webhook creation in Jira, stage 2

Scroll down and click Create. To test the solution, create an issue in the project. A message in Teams appears, as pictured in Figure 16.

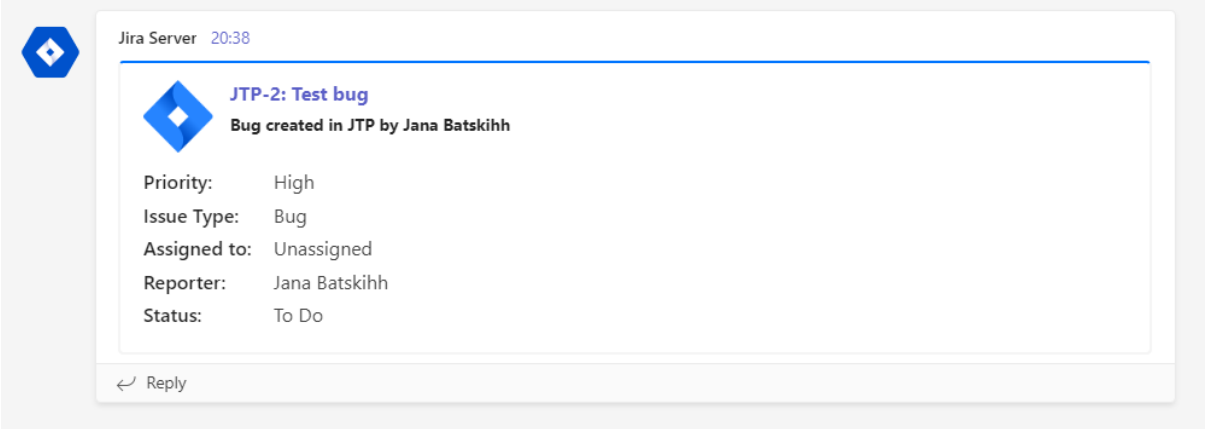


Figure 16. Jira Server notification in Microsoft Teams

There is a possibility to implement a similar solution using non-native functionality with Automation for Jira plugin. While it will not be described here in detail, since it requires setting firewall rules (which is out of my responsibility), however, the idea will be briefly explained. Similarly, a connector should be added to the Teams channel – in this case, *Incoming Webhook*. Just like in the previous solution, it presents the URL that should be used as an integration link between

Teams and Jira. Figure 17 demonstrates the configuration window of Incoming Webhook connector.

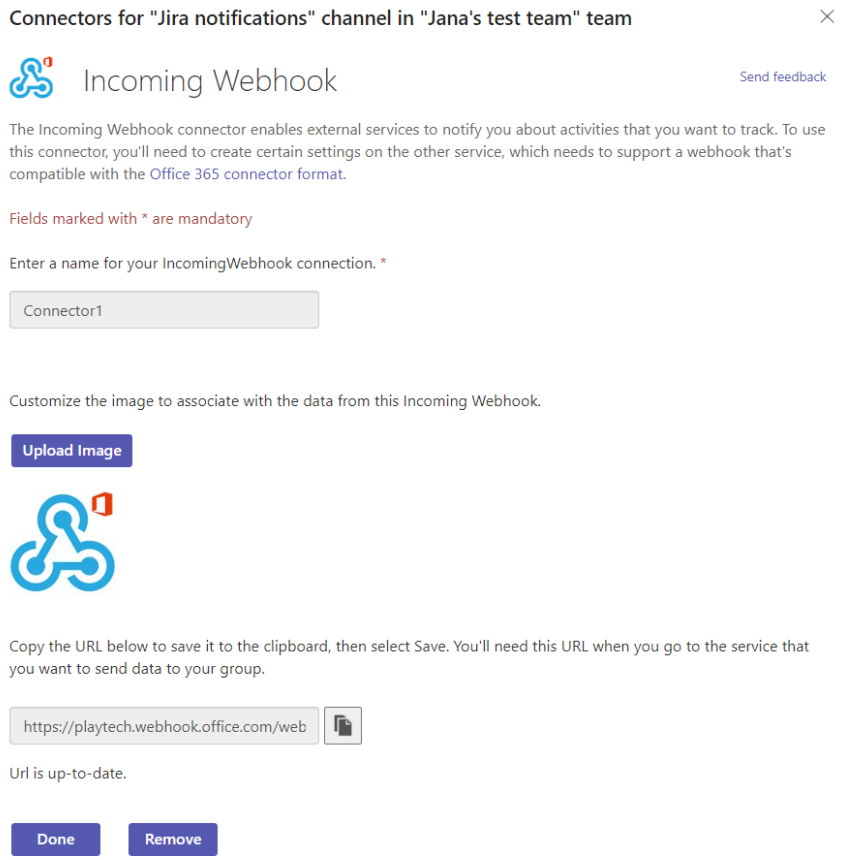


Figure 17. Incoming Webhook Connector in Microsoft Teams

Next, in Jira, locate the Automation for Jira plugin and create a new rule. Similarly, create a trigger on *Issue created* and add a new action – *Send Microsoft Teams message* (Figure 18).

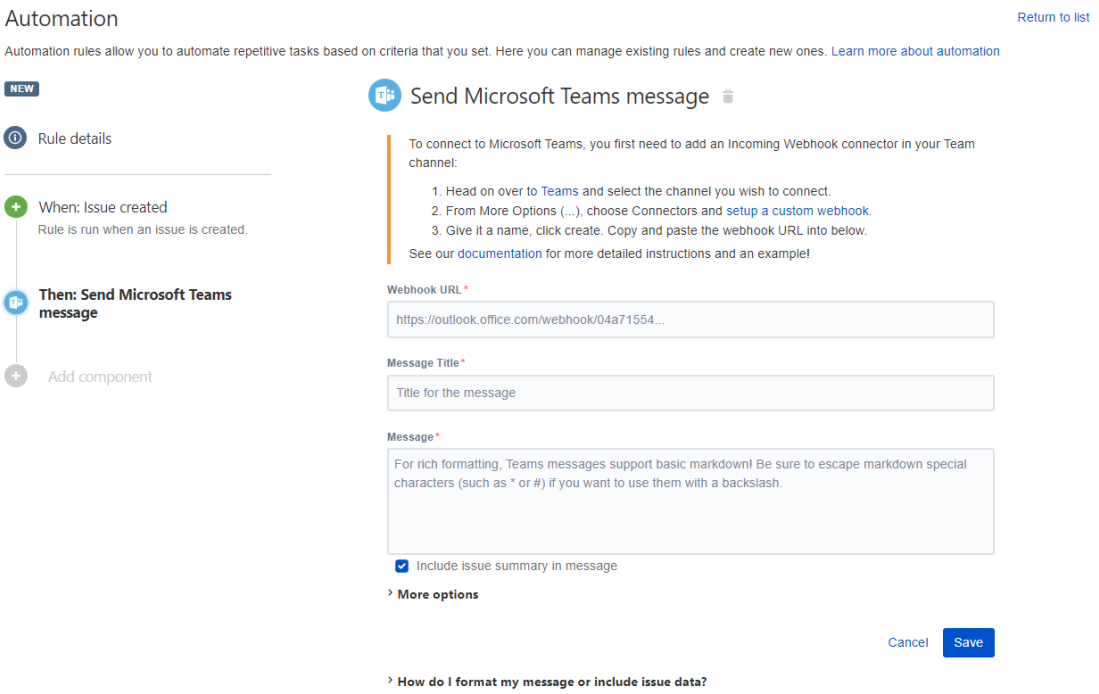


Figure 18. Automation for Jira rule creation, Microsoft Teams notification

Add the URL from the connector, create a message title and the message itself. For the message, you can also use smart values from the first solution, e.g., *{{issue.key}}, {{issue.url}}, {{assignee.displayName}}.*

3.3.2 Automatic sub-task creation

Another way to automize the process of development can be the creation of specified subtasks. For example, users create a Story with a name ‘Game creation’, and then a sub-task is automatically created with Testing as a title. For this solution, a plugin named *ScriptRunner* is used.

Open *Workflow* tab in Project settings. We can see that all issue types use one workflow (Figure 19). To implement the solution, we need to separate the workflow.

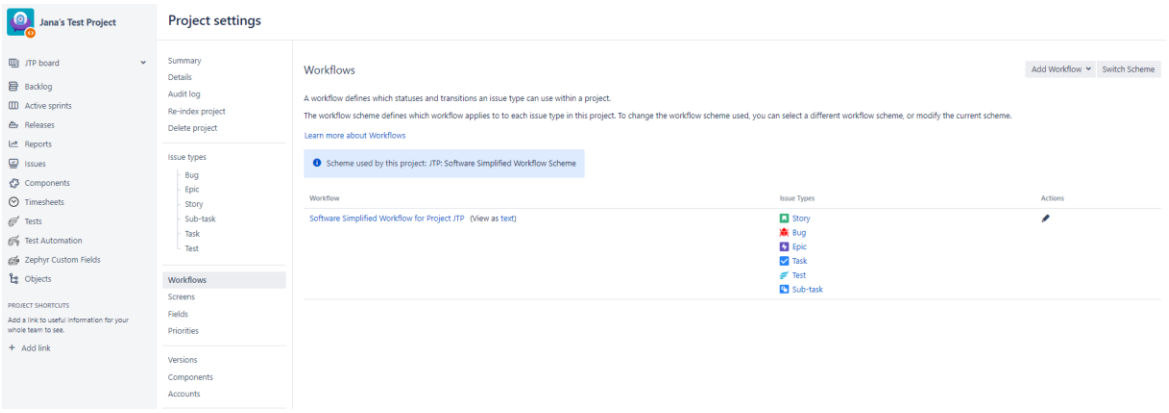


Figure 19. Workflow settings in a Jira project

To separate the workflow and assign a new one to Story issue type, press full stop (.) while located on the page, and type ‘workflow’, as seen in Figure 20.

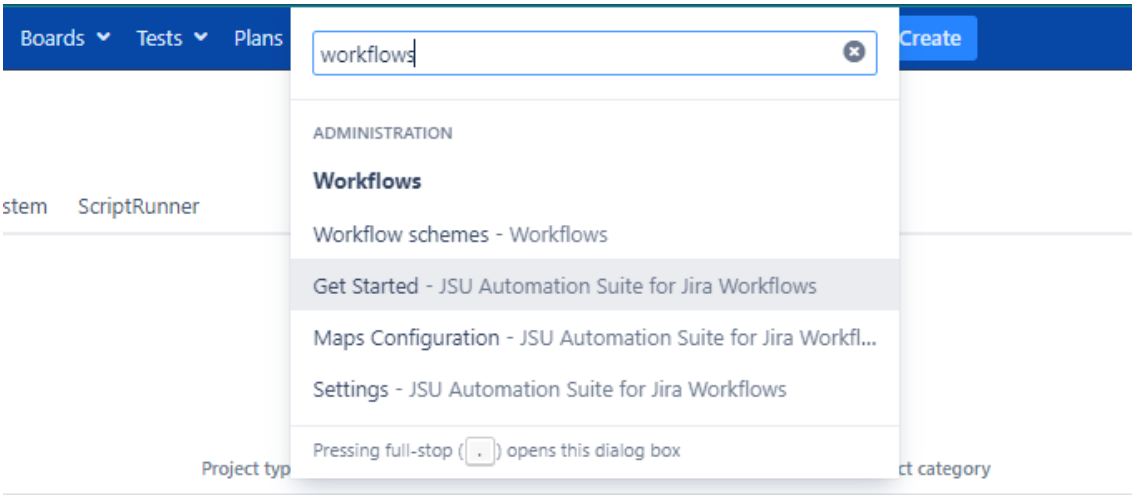


Figure 20. Pop-up search window in Jira

Then press Enter to move to Workflow page. Press the combination of Ctrl+F to open a find box. Copy the name of the project workflow and paste it into a box, then press Enter to locate it (Figure 21).

Figure 21. Project workflow in the list

Click on *Copy* to create a copy of this workflow. Choose a name for it and when done, click *Copy*, as seen in Figure 22.

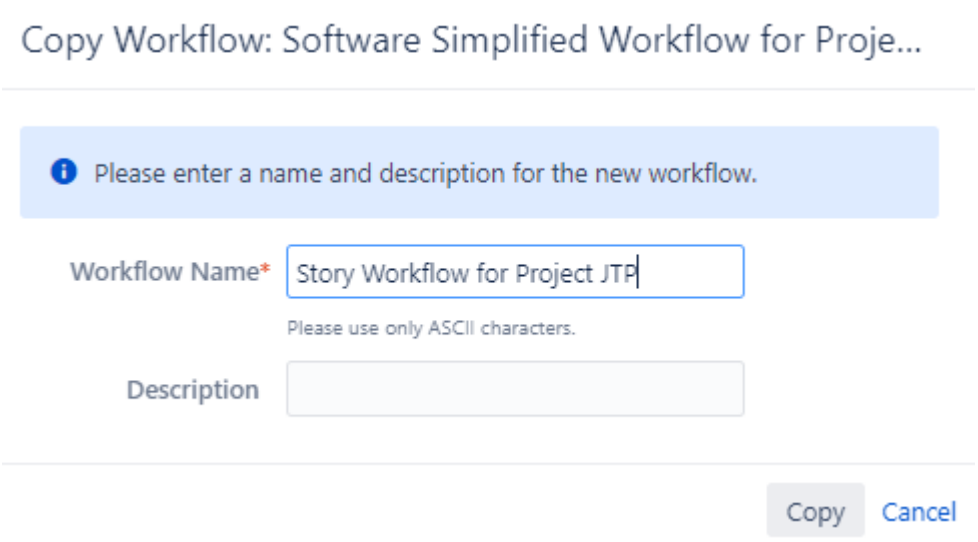


Figure 22. Copy workflow window

Newly copied workflow will automatically load up. Click on *Diagram* to change the view from *Text*. The workflow should be visible as demonstrated in Figure 23.

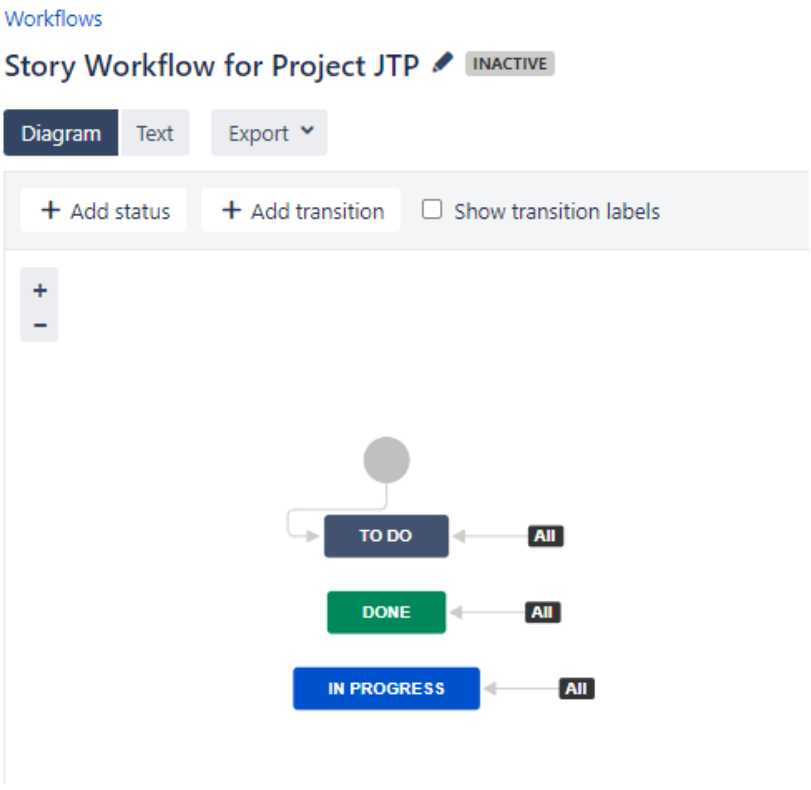


Figure 23. Workflow (diagram view)

Next, click on a transition line which goes from the circle to *To Do* status. This transition line represents create event. A window on the right side of the workflow appears (Figure 24).

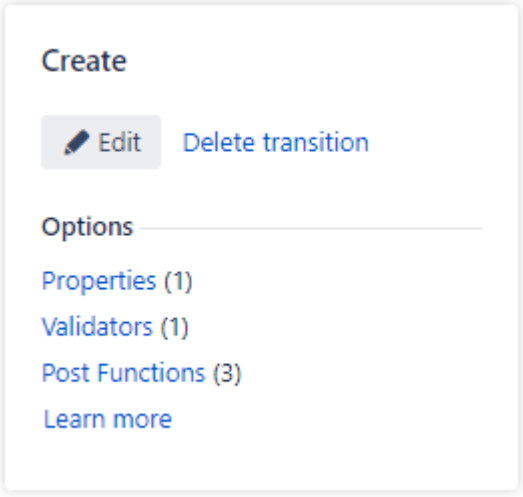


Figure 24. Create workflow transition settings

For the solution, we are going to implement a *post function*. A post function refers to a workflow function that carries out automated actions once an issue moves to a different status (Adaptavist, 2023). Click on *Add post punction*, as illustrated in Figure 25.

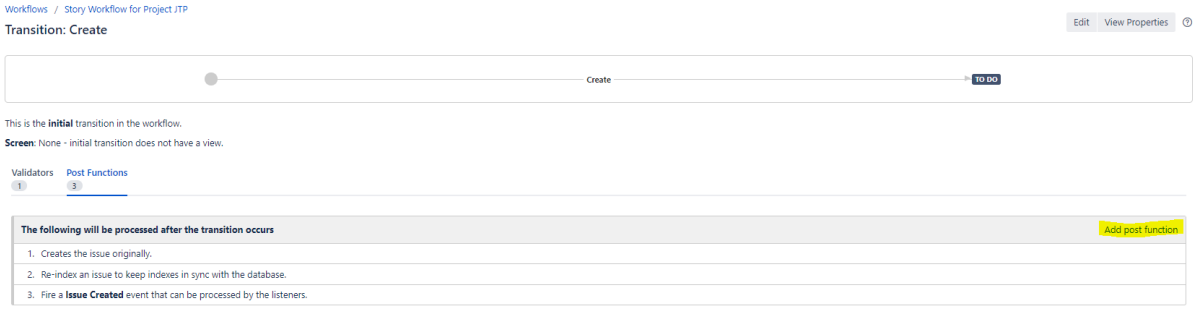


Figure 25. Post function section in a workflow transition

In the list, locate *Create a sub-task [ScriptRunner]* and select it (Figure 26).

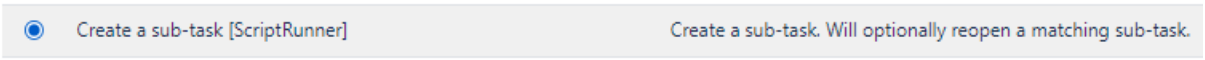


Figure 26. ScriptRunner post function selection

Scroll down and click *Add*. A *ScriptRunner* menu appears (Figure 27).

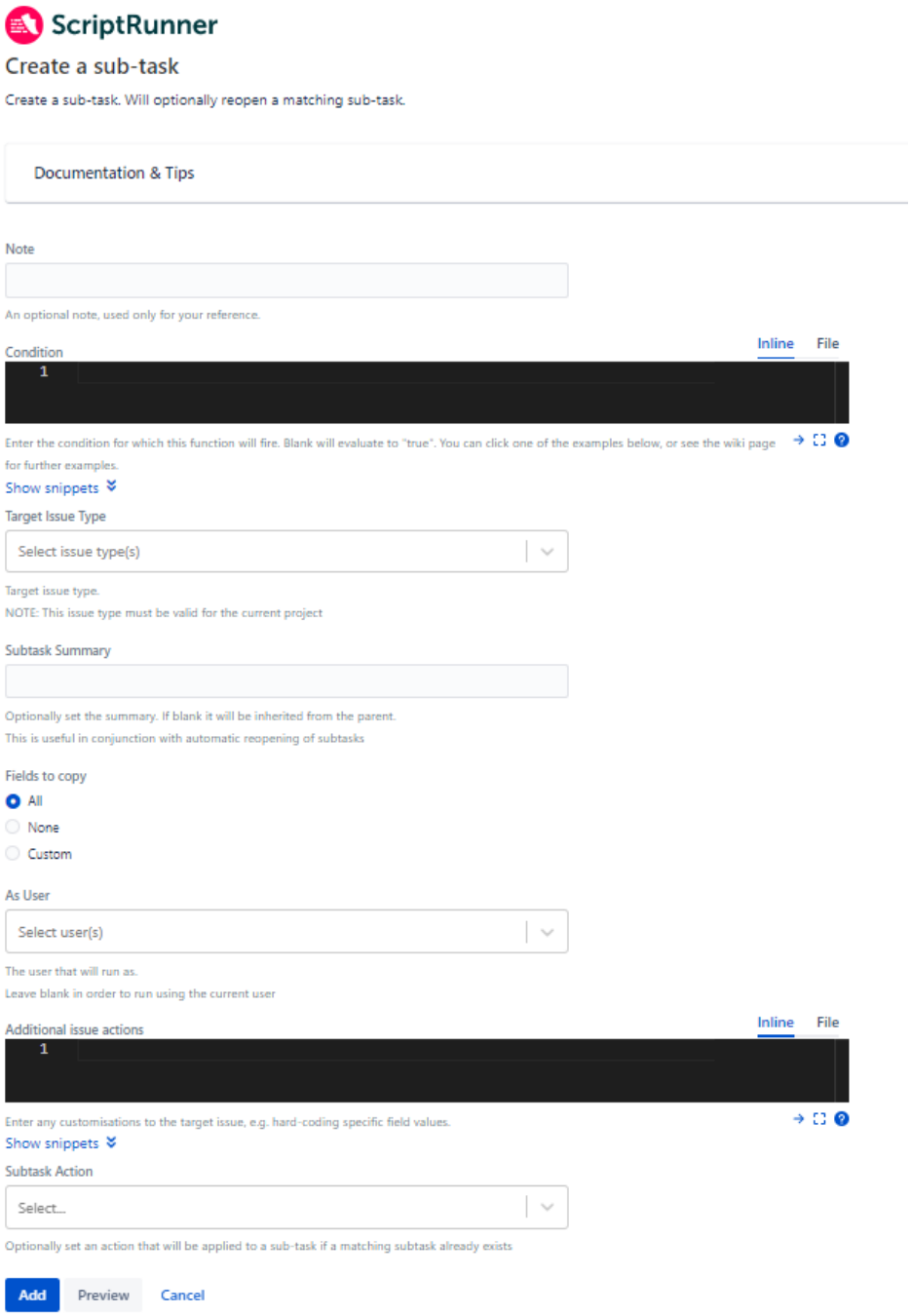


Figure 27. ScriptRunner Create a sub-task post function menu

Give a name to the automation and write it into the *Note* box. *Condition* will be left blank because we want all Stories to create a sub-task. Select *Target Issue Type* to Sub-task (verify that the project uses this issue type). Leave *Subtask Summary* blank, as we will compose the summary using Groovy script in the box below. For *Fields to copy*, choose *Custom* and add Priority and Reporter – the sub-task will have the same values as the parent issue. For the user, choose Jira Automation

(or any other dummy user set for automation purposes). Leave *Subtask Action* empty.

For the *Additional issue actions*, we are going to create an automatic summary that consists of Testing + parent issue key. Additionally, we are going to add a script that will show a popup message that appears once the automation is successful. The script is seen in Figure 28.

```
issue.summary = 'Testing - ' + sourceIssue.key
import com.onresolve.scriptrunner.runner.util.UserMessageUtil

UserMessageUtil.success('The sub-task is successfully created.')
```

Figure 28. Groovy script for sub-task automation

Final look of the *Create a sub-task* menu is demonstrated in Figure 29.

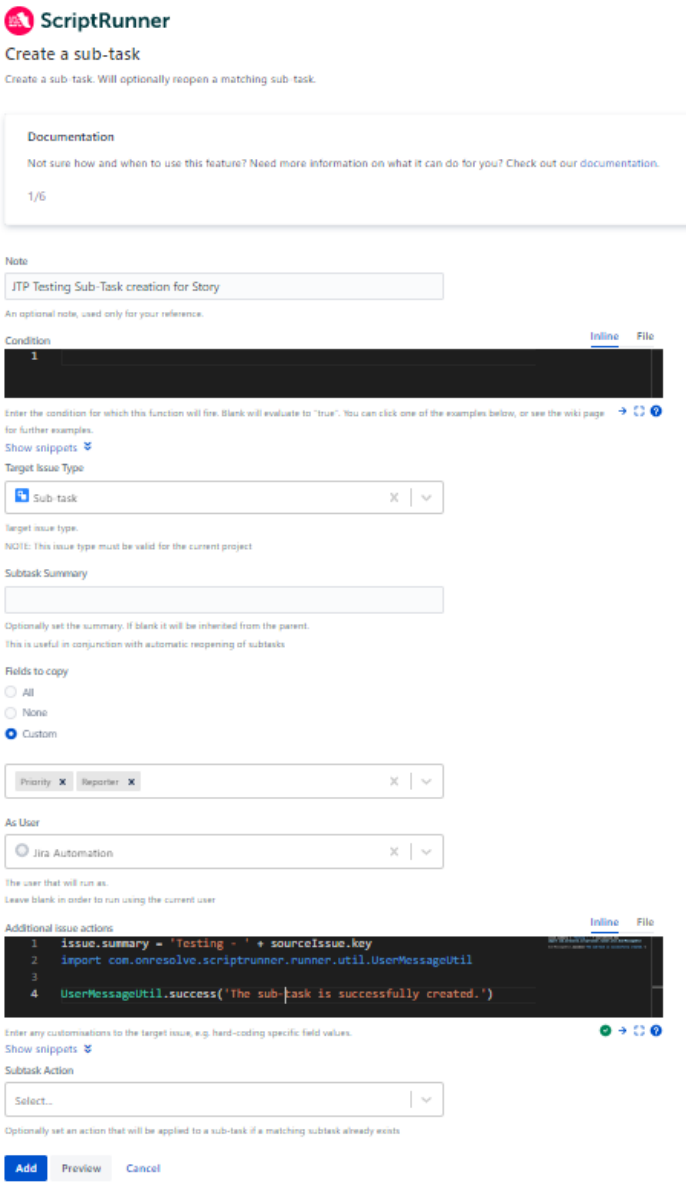


Figure 29. ScriptRunner Create a sub-task menu (final view)

Once happy with the settings, click on *Add*. Be sure to place the post function after the first step. Use the arrows on the right to modify the position of the post function. Figure 30 illustrates the result.

Figure 30. Post function list with newly created post function

To apply the new solution, the workflow should be added to the workflow scheme of the project. To do this, press full stop (.), begin writing ‘workflow’ and select

Workflow schemes. Once the page loads up, find the workflow our project uses (easiest way is to press Ctrl+F and type JTP – the project key). Click on *Edit* to modify the settings, as shown in Figure 31.

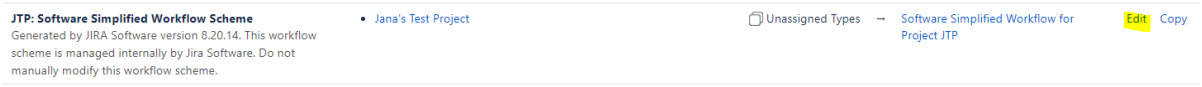


Figure 31. Workflow scheme to modify

Click on *Add Workflow -> Add Existing*. In a popup window (Figure 32), find the workflow we have recently modified using the search. Once it is found, click *Next*.

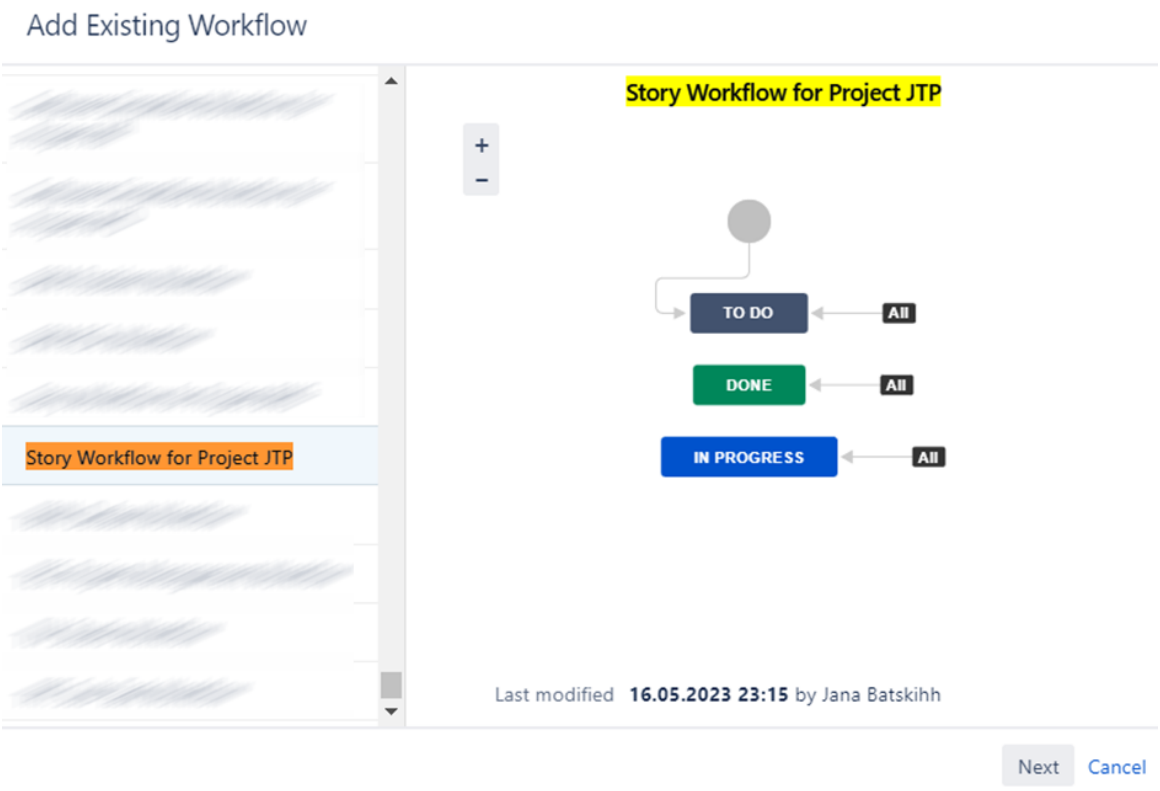


Figure 32. Add Existing Workflow window

Find *Story* issue type (Figure 33), select it and click *Finish*.

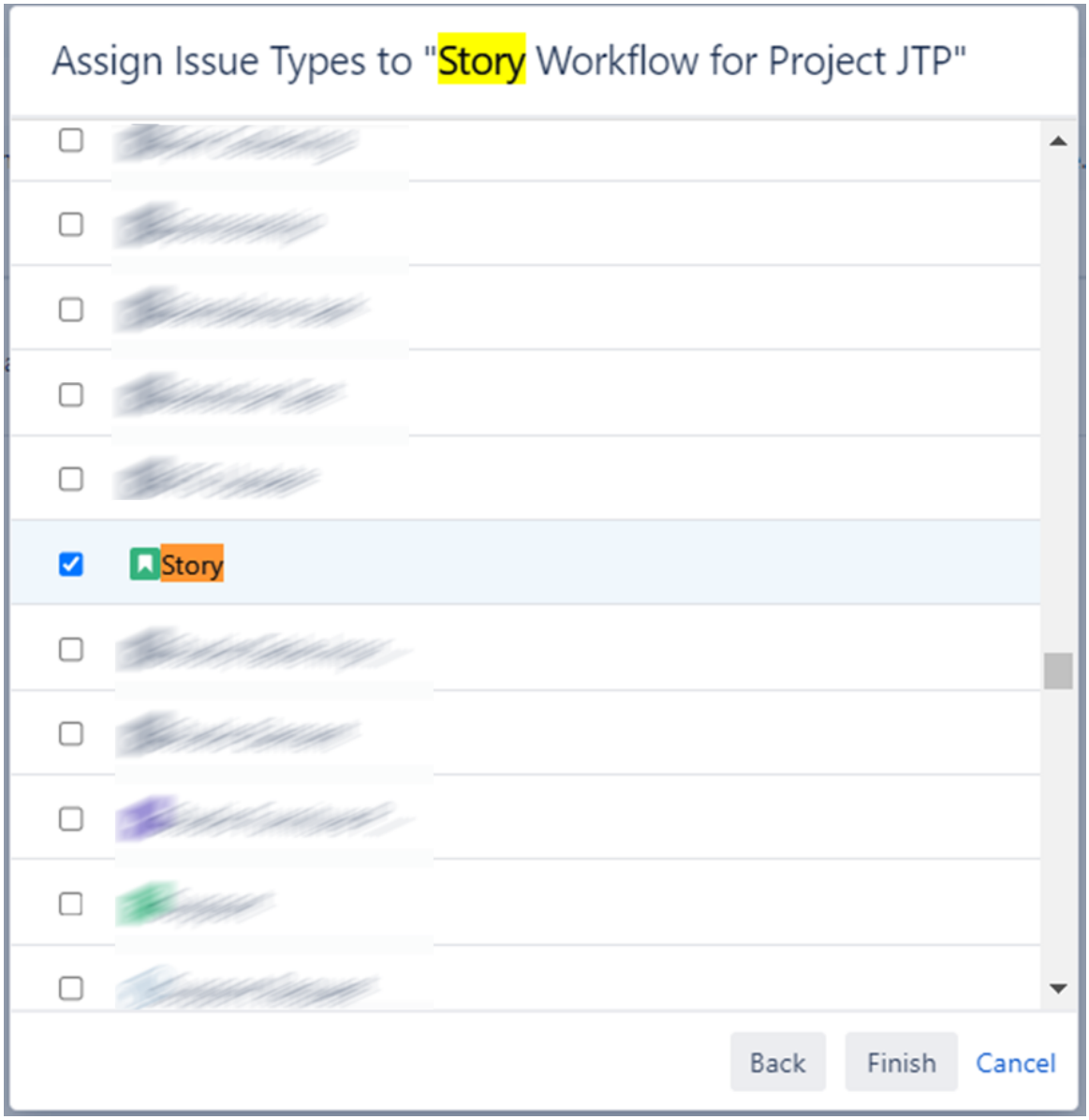


Figure 33. Assign Issue Types to workflow window

Once done, click on Publish button. Afterwards, click Associate. Once the migration of issues to the new workflow finishes, click Acknowledge. Return to the project and observe the changes (Figure 34).

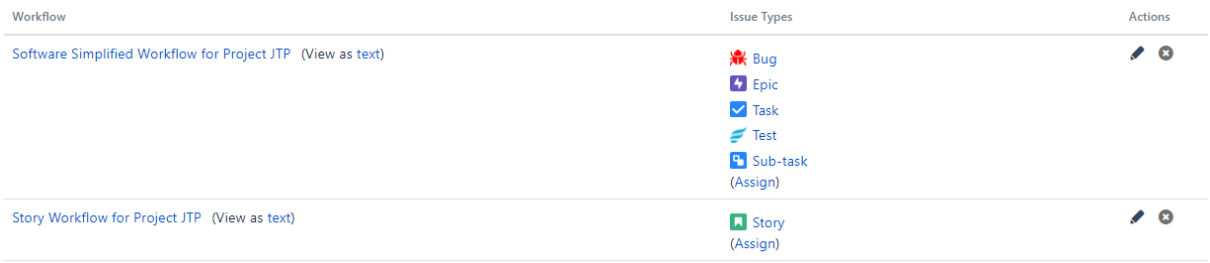


Figure 34. New workflow settings page in the project

To test the solution, create a new issue. Remember to choose Story and additionally modify the Priority to High. When finished, click Create. Observe the newly created issue; the final view should look like Figure 35.

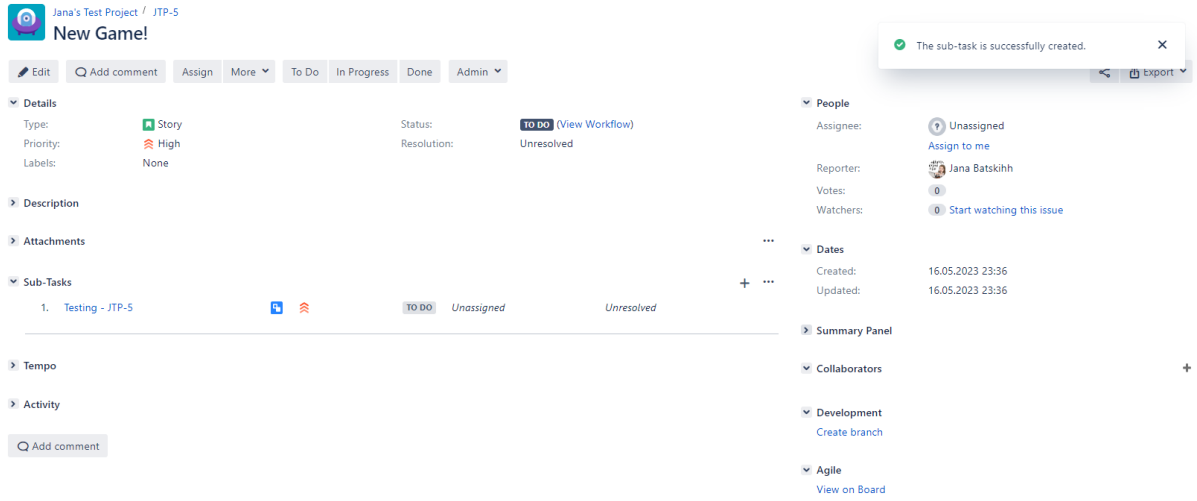


Figure 35. Final result of automatic sub-task creation automation: Story (parent page)

As one can see in Figure 36, the sub-task is created as expected, and the popup message we have set up also appears. Notice that the sub-task also has High priority as the parent issue. Reporter in sub-task is also the same as in parent issue.

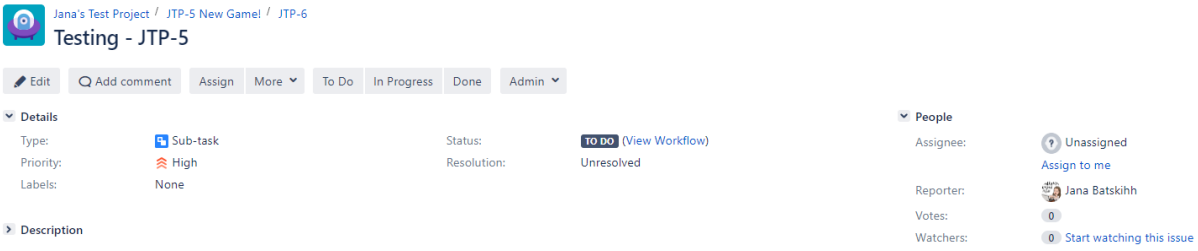


Figure 36. Final result of automatic sub-task creation automation: Sub-task (child page)

Additionally, other sub-tasks can be added using the similar method. For example, creating additional sub-task called *Documentation*.

3.3.3 Pre-set description of an issue

In this case, we are going to review a different scenario that has also proven to be beneficial in many cases. Suppose that the team wants to have a pre-set description field to speed up the process of their work. This can be done, for

instance, in reporting bugs. The solution can be achieved using *ScriptRunner* plugin.

Type full stop (.) and write *ScriptRunner*. Select *Browse ScriptRunner*. Go to *Behaviours* tab and click *Create Behaviour*. Destination page should look like in Figure 37.

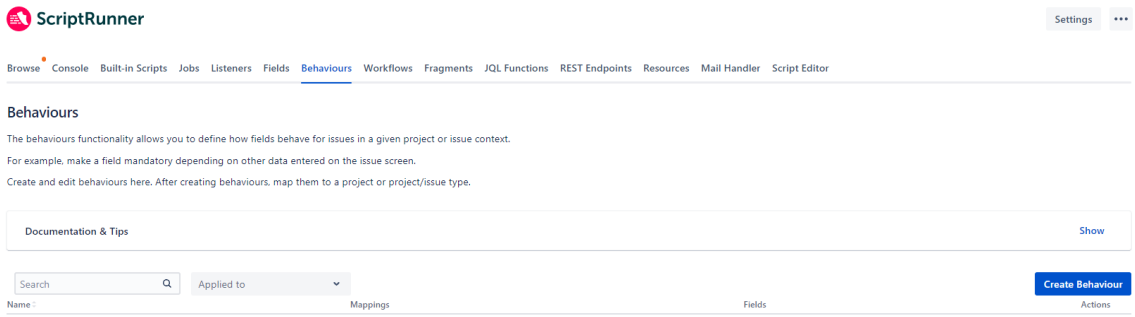


Figure 37. ScriptRunner Behaviours page

Once a popup window appears, click on *Create Mapping*. Give a name to the behavior, choose the project and issue type scope. In the end, page should look like Figure 38 demonstrates.

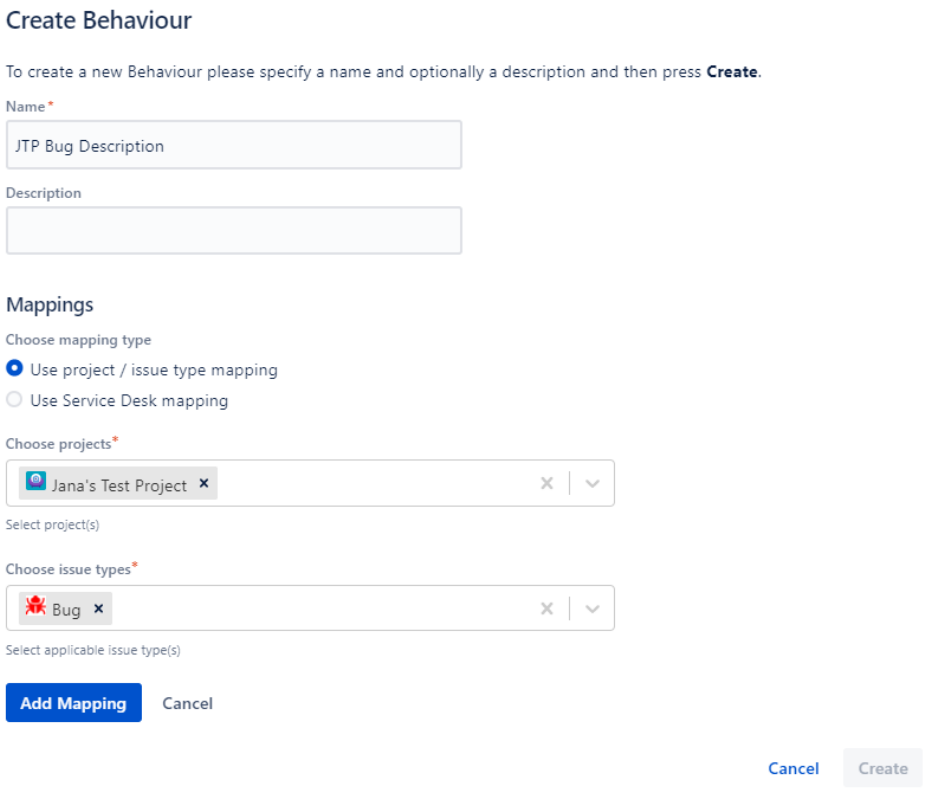


Figure 38. Create Behaviour window with name and mapping entered

Click on *Add Mapping* and then *Create*. When a page loads up, click on Create Script. A box to write code appears. Suppose that we want to use the description as shown in Figure 39. Note that the text is in bold.

Environment:
Instance:
Regulation:
User/ Password:
Game Name:

Steps to reproduce:
1.
2.
3.

Actual Result:
Screenshot:
Expected Result:

Figure 39. Text to be added to issue description

To write the text in bold, asterisks should be used before and after each row. The Groovy script used in this solution is demonstrated in Figure 40.

```
def desc = getFieldById("description")

def defaultValue = """*Environment:*
*Instance:*
*Regulation:*
*User/ Password:*
*Game Name:*

*Steps to reproduce:*
*1.*
*2.*
*3.*

*Actual Result:*
*Screenshot:*
*Expected Result:*

""".replaceAll(/    /, '')

if (! underlyingIssue?.description) {
    desc.setFormValue(defaultValue)
}
```

Figure 40. Script used for pre-populating the description field

Leave other fields on default. Once done, click *Save changes*. To test the solution, click on *Create* button in the Jira page header and select *Bug*, as illustrated in Figure 41.

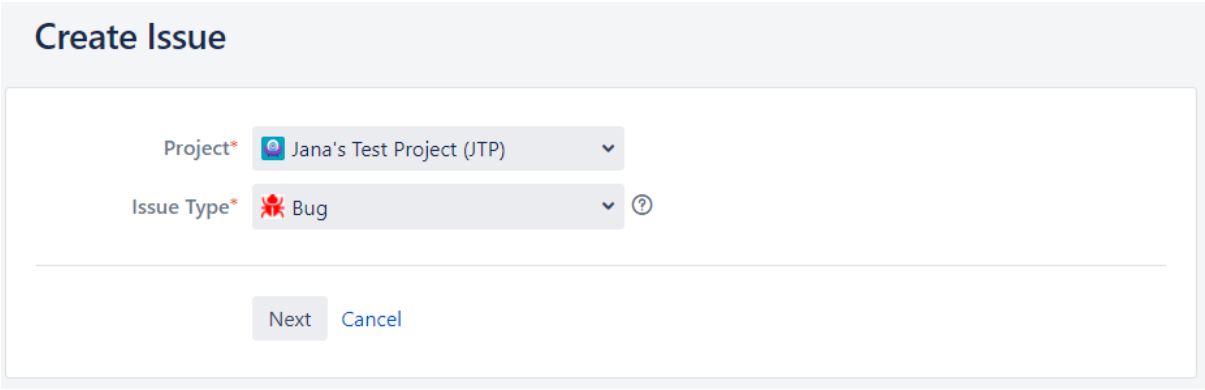


Figure 41. Create Issue window with Bug as an issue type

After clicking *Next*, observe the *Description* field. Figure 42 illustrates that it is already pre-filled with the description we have set in the behavior automatization.

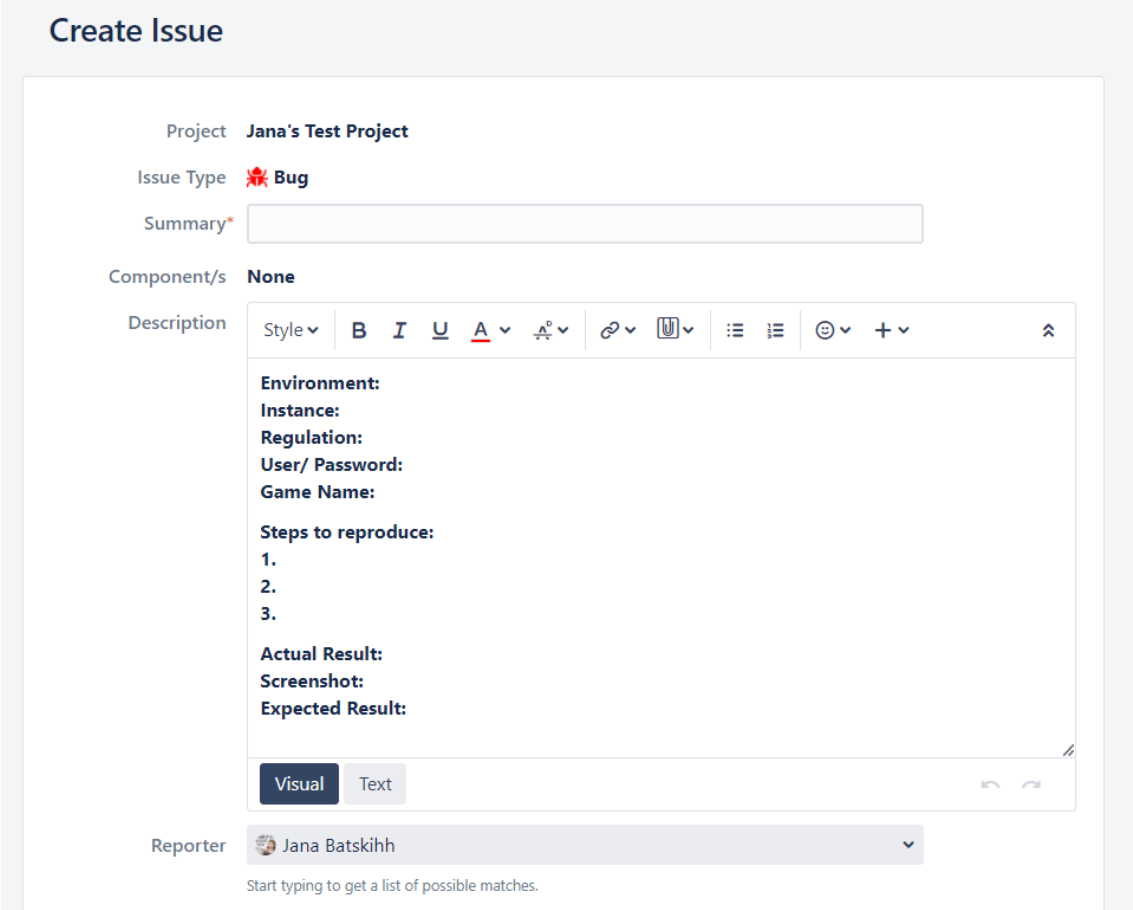


Figure 42. Pre-filled description field on Create screen

Similarly, other automatic descriptions can be set up, based on the needs of a team.

3.3.4 Auto-assignment to QA user

Suppose that we want to modify the workflow transition from *In Testing* to automatically include a user in a QA role. Doing so will save time for users, eliminating the need of manually checking Jira. In this case, we are going to use ScriptRunner functionality to modify the workflow.

For this task, a new issue type called New Feature is added to the project. It uses the workflow pictured in Figure 43.

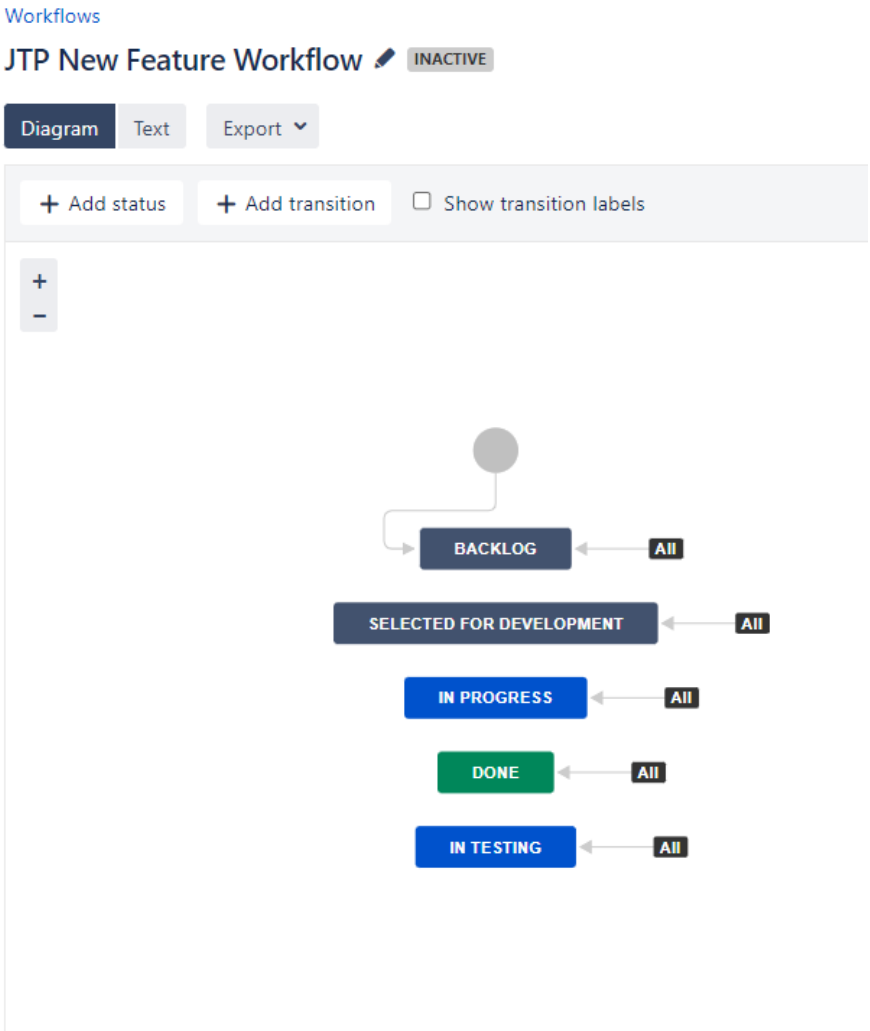


Figure 43. New Feature issue type workflow in Diagram mode

The workflow is set so that users can move issue from one status to any other status. Modify the issue type scheme in the project to include a new issue type. Click *Actions -> Edit issue types*, as shown in Figure 44.

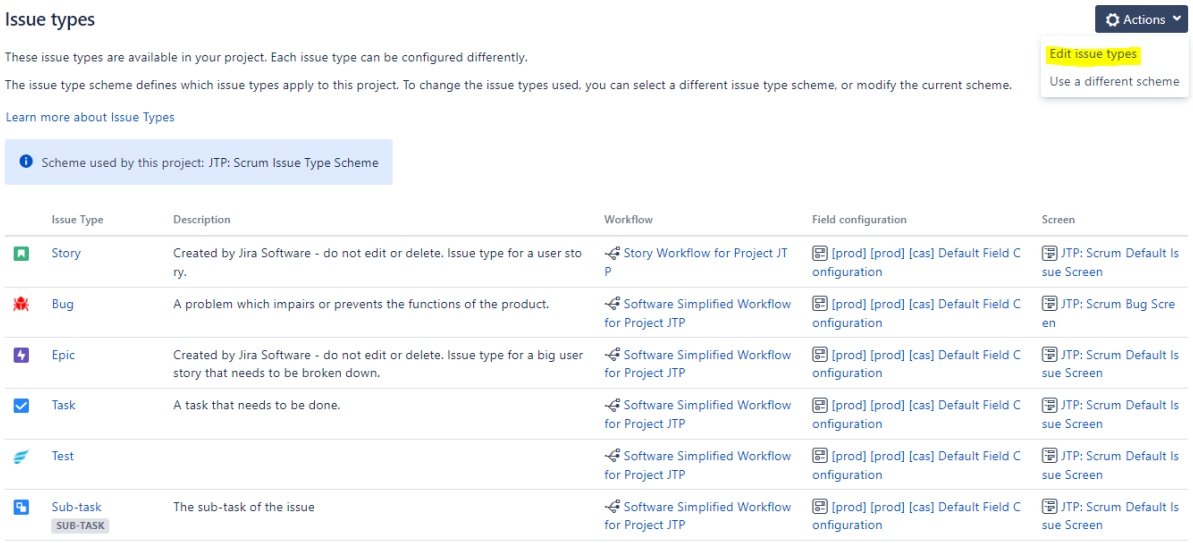


Figure 44. Issue types settings page

In the list, add the new issue type to the project. Next, go to the *Workflows* tab in the project settings to modify workflow of New Feature (note that it has been automatically assigned to default workflow in the project). Figure 45 illustrates how to add a new workflow to the project. Click *Add Workflow* and then *Add Existing*.

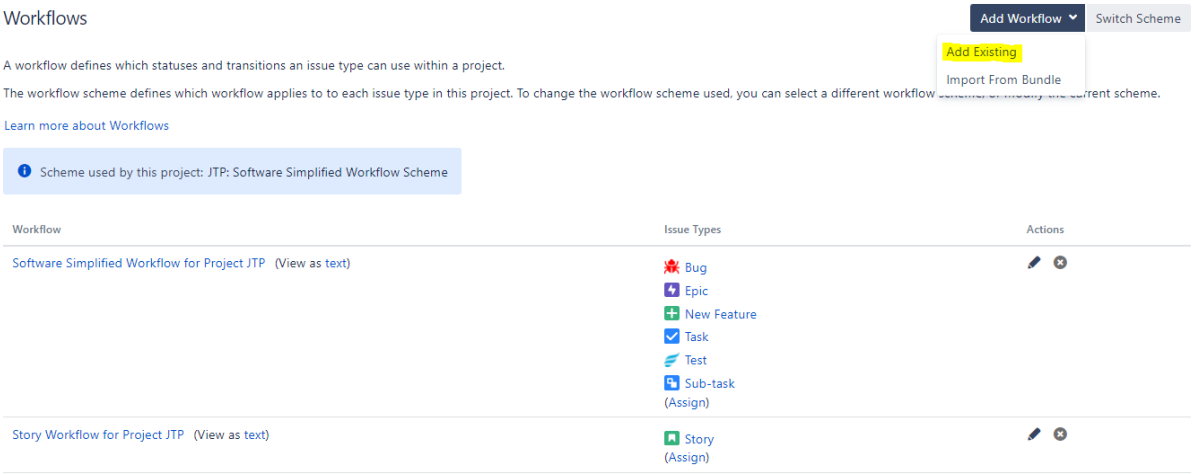


Figure 45. Workflow settings page with newly added issue type

Find the workflow by name to add it to the project. As shown in Figure 46, assign the new issue type to the new workflow and click *Finish*.

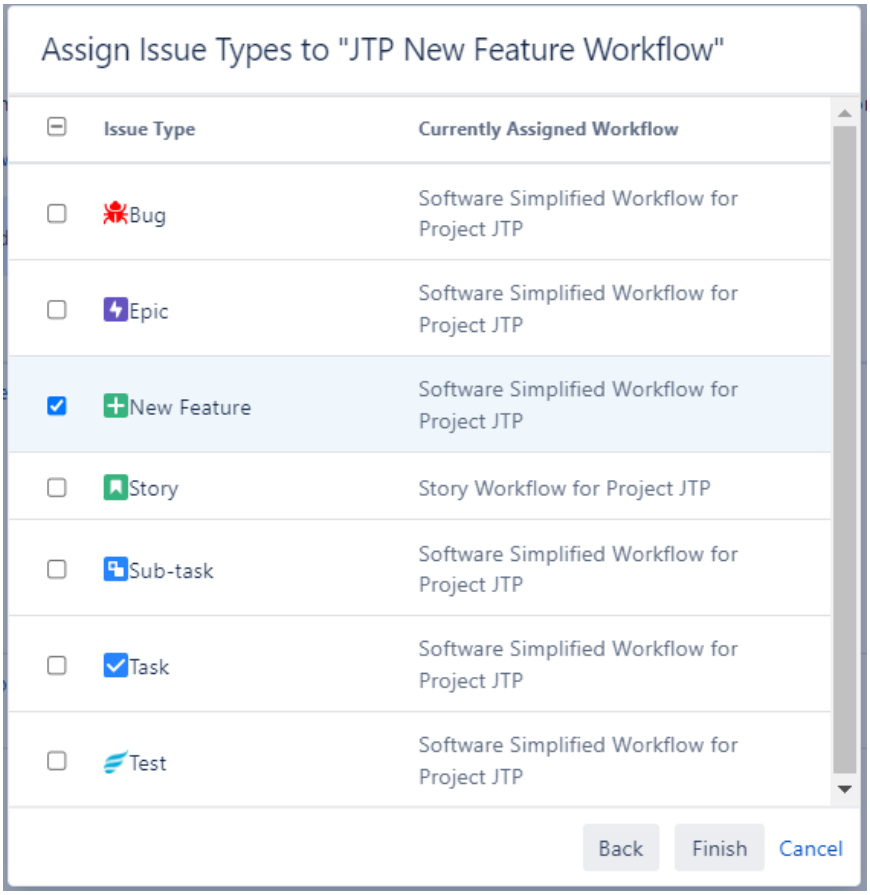


Figure 46. Assigning the workflow to the issue type

Do not forget to click *Publish* after adding the workflow. Now after the new issue type and workflow have been added to the project, it is now possible to implement the solution. In the new workflow, click on *Edit*. Then, find the *In Testing* status and the transition arrow to it. Select it and click on post function to add a new functionality. Click on *Add post function* and locate *Assign to first member of role [ScriptRunner].* The final page is illustrated in Figure 47.

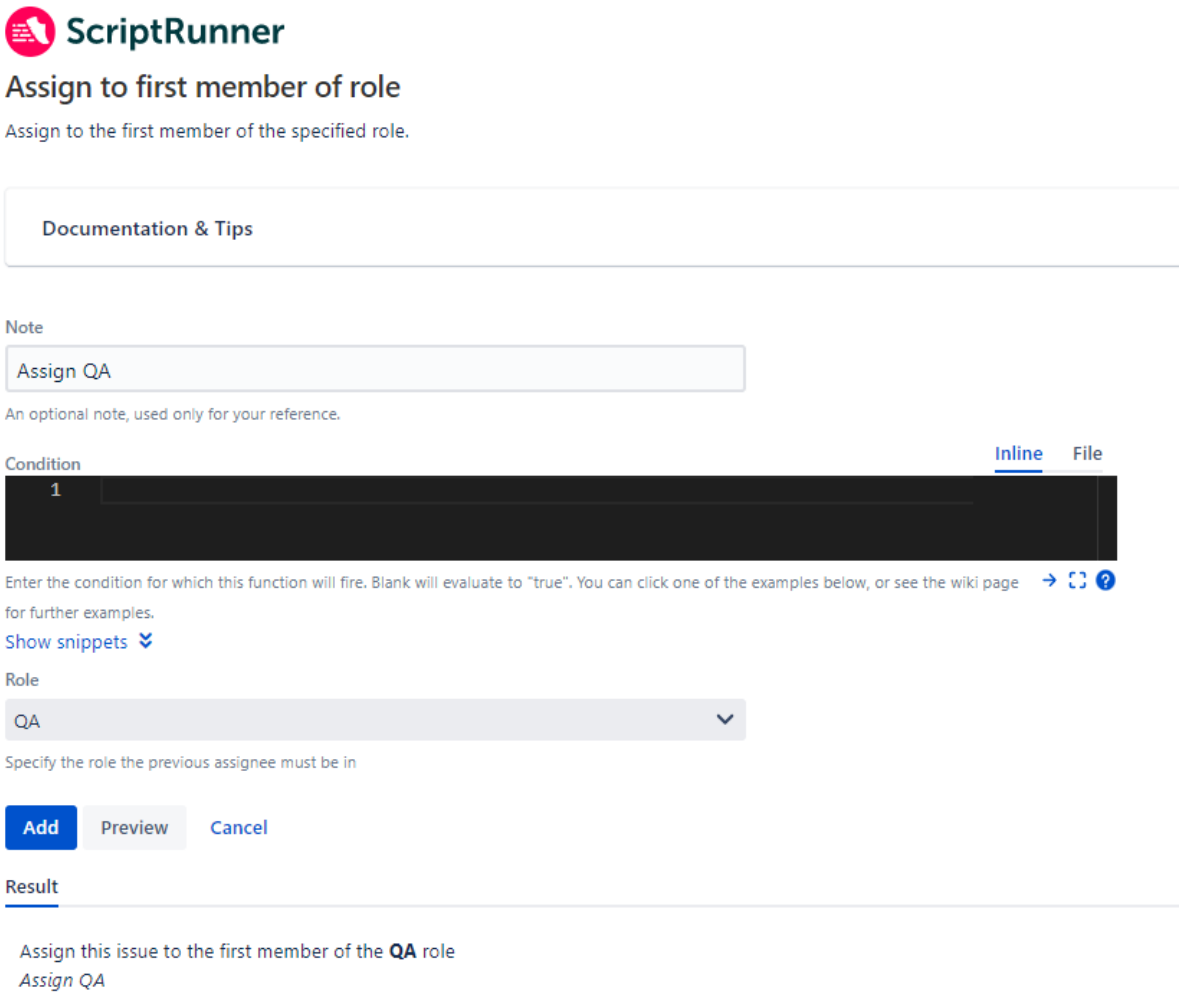


Figure 47. Assign to first member of role post function menu

In the Role section, find QA and select it. Everything else can be left blank. Click *Add* once done. When the page loads up, same post functions should be seen as in Figure 48. Save changes and publish the workflow.

The following will be processed after the transition occurs

1. ScriptRunner workflow function - Assign this issue to the first member of the **QA** role
   Assign QA
2. Set issue status to the linked status of the destination workflow step.
3. Add a comment to an issue if one is entered during a transition.
4. Update change history for an issue and store the issue in the database.
5. Re-index an issue to keep indexes in sync with the database.
6. Fire a **Generic Event** event that can be processed by the listeners.

Figure 48. Newly created post function in the list

For the next step, in project settings, go to *Users and roles*. Click *Add users to a role,* choose the person and specify the role (Figure 49).

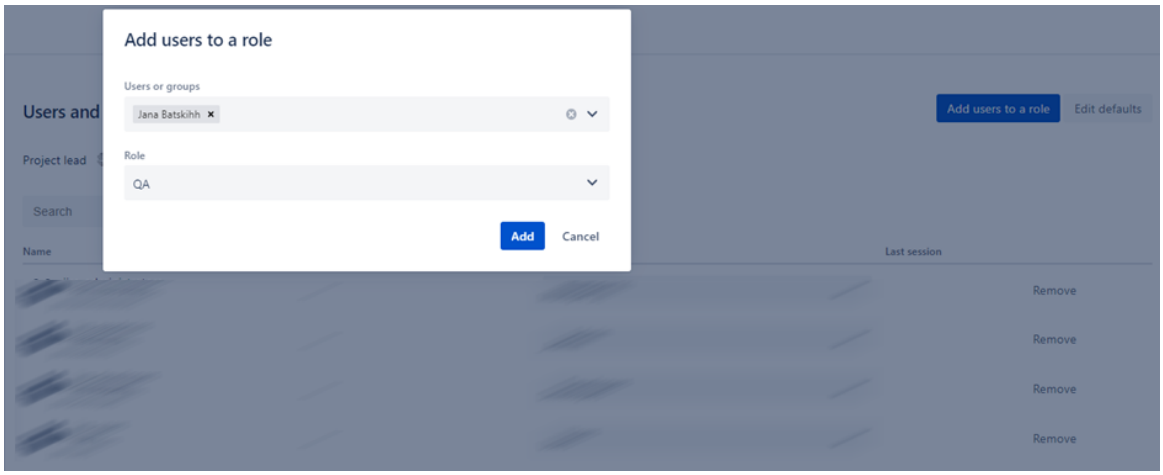


Figure 49. Adding a new QA role

Test the solution by creating a New Feature issue. Once created, assign the issue to dummy user (e.g. Jira Automation) and move to any other transition other than *In Testing* (optionally). The assignee should stay the same. Finally, move the issue to *In Testing* and observe the changes – the QA user should be automatically assigned to the issue. While it is not reflected in the history of the issue (Figure 50), because the automation sets the user who made the transition to *In Testing* as the user who changed the assignee. Nevertheless, the automation was successful.

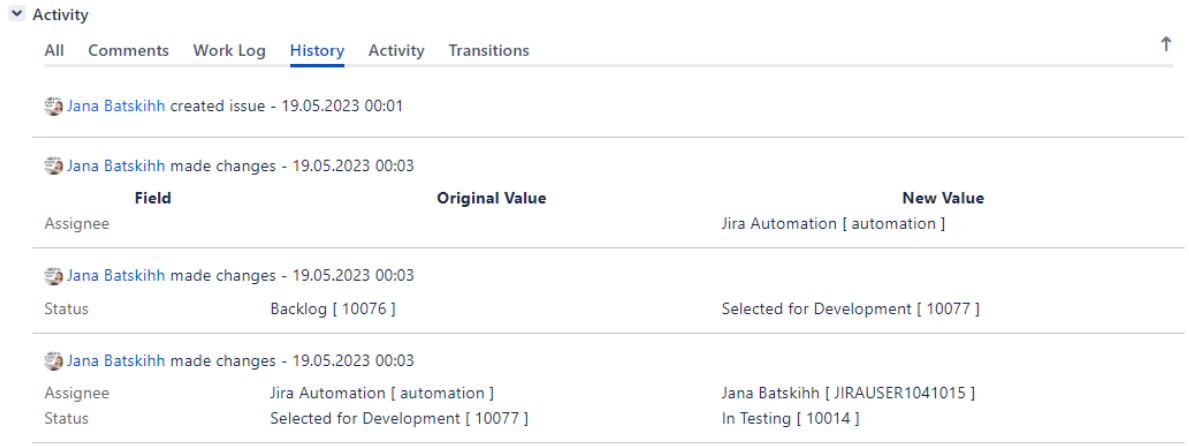


Figure 50. History of the issue, part 1

Moreover, when the issue gets unassigned and goes from any other status, for example, In Progress, it still correctly transitions and reassigns the correct person from QA (Figure 51).

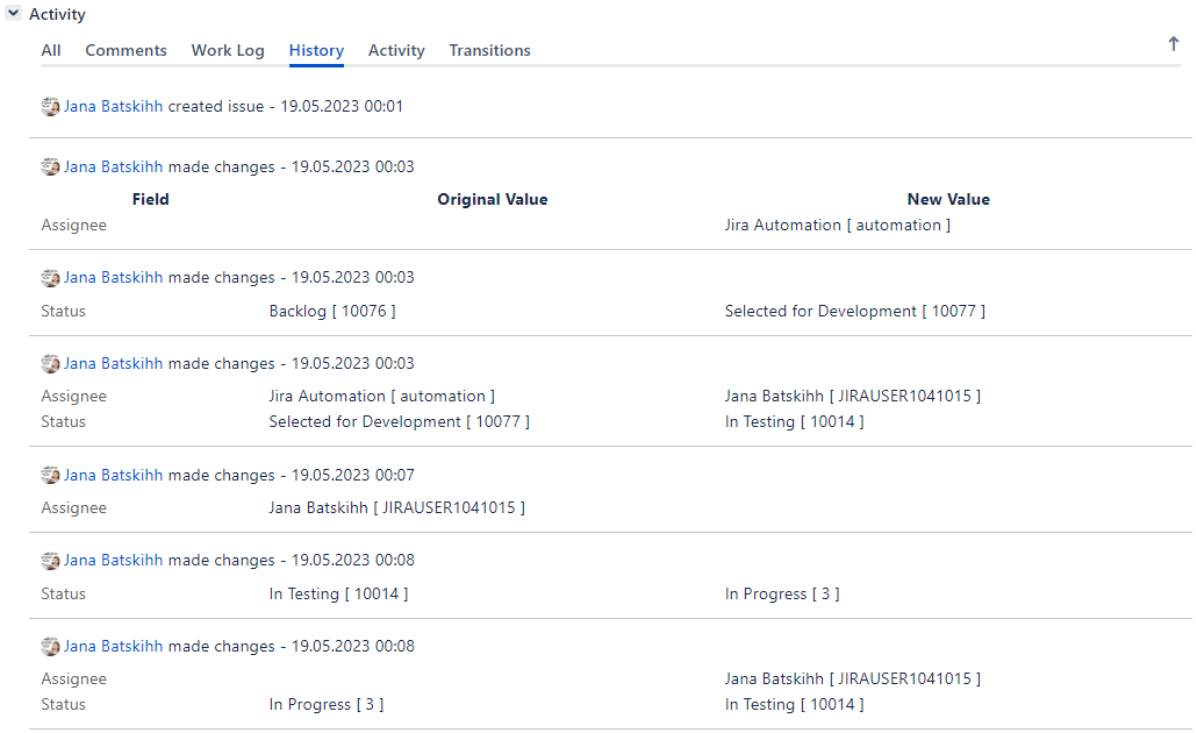


Figure 51. History of the issue, part 2

Additionally, the automation can be set up differently. For example, to include Team Lead on a specific status (e.g., Team Lead review). By default, Jira will also modify the user that the issue is assigned to them, skipping the time to check manually for updates. It is also possible to auto-assign not to the role, but

to the specific user. Unfortunately, the automation does not work for roles with multiple users, since the Assignee field requires only one user to be selected.

3.3.5 Restricting transitions: condition blocking from closing issue

In this scenario, our aim is to enforce a solution that prevents users from closing issues that have incomplete sub-tasks. Although it does not directly accelerate users' work, it aids in task organization and prevents the premature accidental closure of ongoing issues. ScriptRunner is utilized in this case to implement the solution.

In Jira, return to the workflow of New Feature. Select the arrow that indicates the transition to Done status. Click on Conditions, as indicated in the Figure 52.

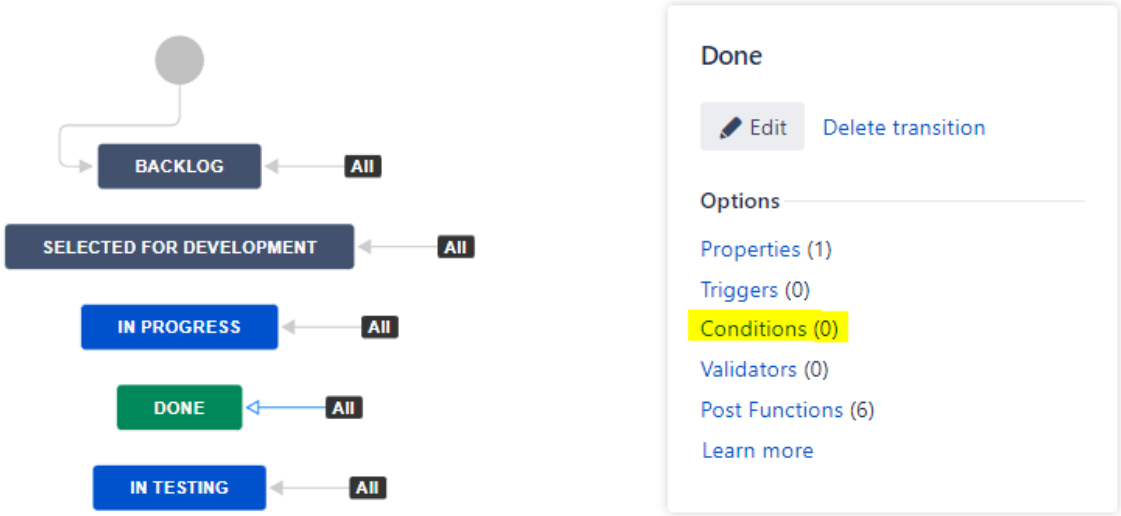


Figure 52. Workflow with Done transition selected

After the page loads, click on *Add condition*. Choose *All sub-tasks must be resolved [ScriptRunner]* and click *Add*. Next, we need to choose Resolution. The Resolution field holds significance within Jira as a crucial feature. It serves to indicate the reason behind closing an issue, eliminating the necessity for multiple statuses solely dedicated to expressing closure reasons (Atlassian, 2023). In this case, as shown in the Figure 53, *Resolution* will be set to *Done* and any other resolutions are not allowed. There is also a possibility to add the same resolution as the parent task.

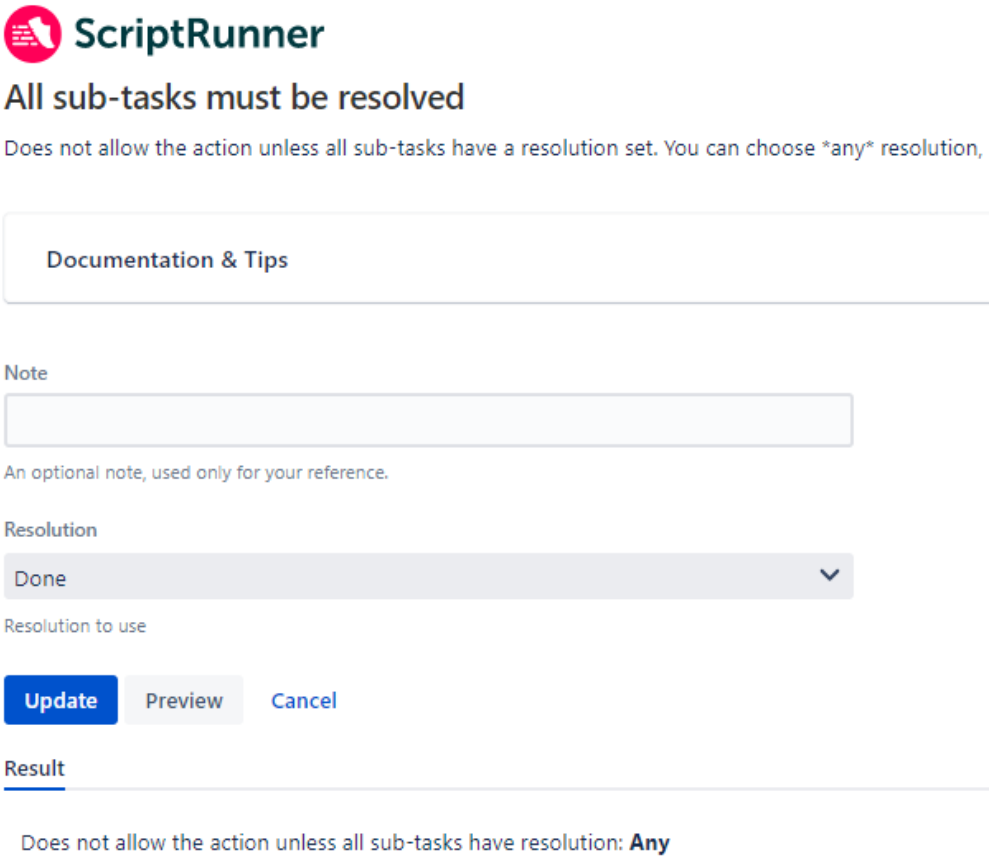


Figure 53. All sub-tasks must be resolved Condition creation menu

Click on *Update* and then publish the workflow. To test the solution, create New Feature issue. Then, on the issue page, find *More*, click on it and find *Create sub-task*, as indicated in Figure 54.

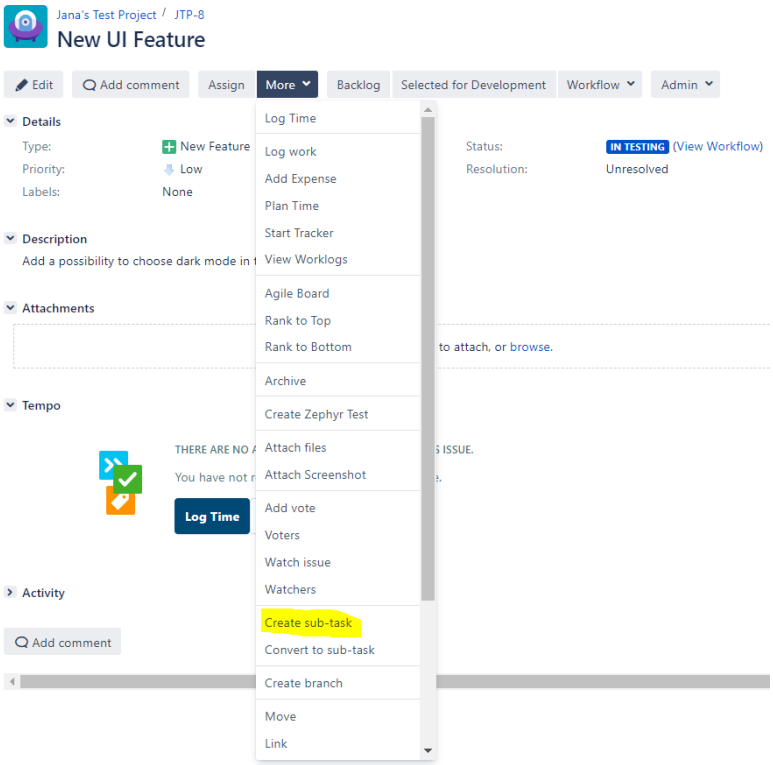


Figure 54. Create sub-task from More menu in the issue view screen

Add a sub-task, tick the box create another, and press *Alt+s* to quickly create the issue. An example sub-task can be seen in Figure 55.

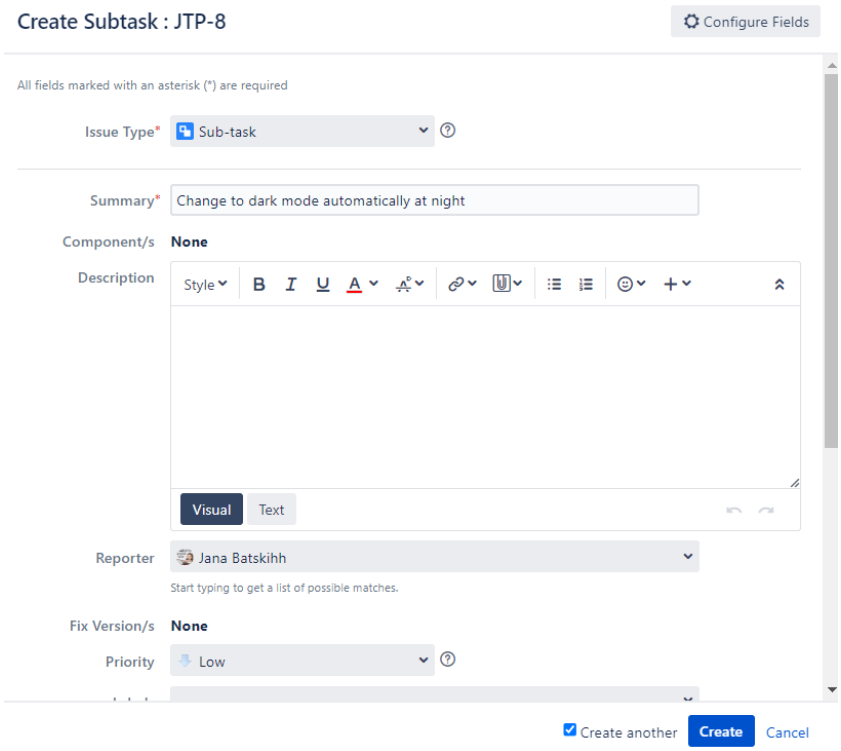


Figure 55. Creating a sub-task popup window

Once both sub-tasks are created, notice that both of them are in *To Do* status (Figure 56). When clicking on *Workflow* button to transition the status to Done, notice that there is no Done status.

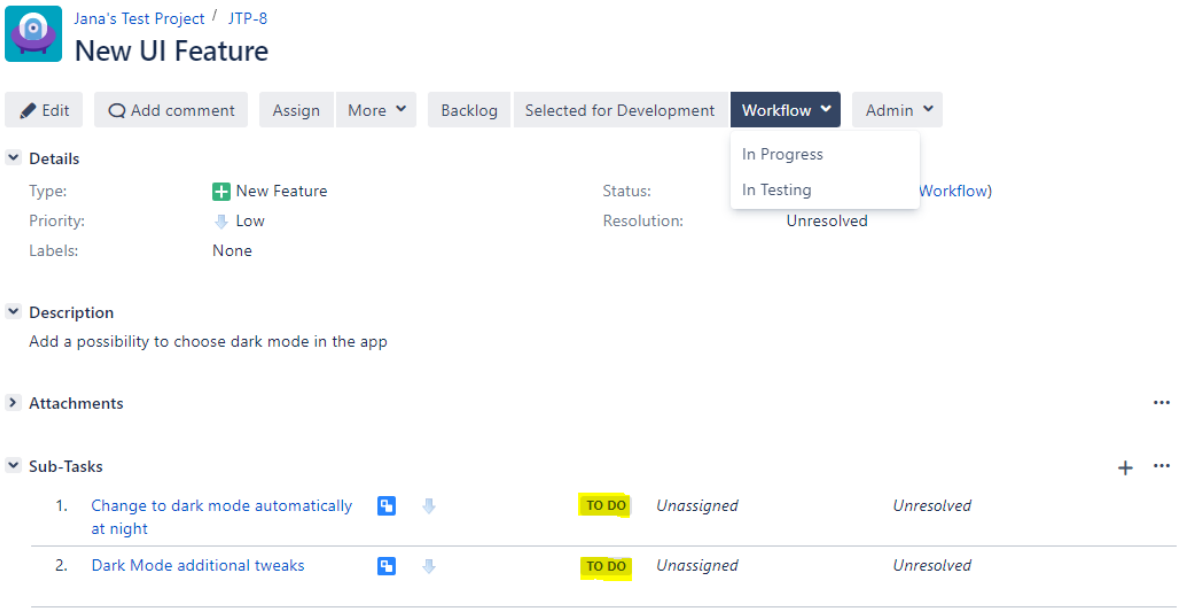


Figure 56. Unclosed sub-tasks and parent issue

Move the both sub-tasks to Done status and return to the parent task. The transition is now visible and can be implemented, as seen in Figure 57.

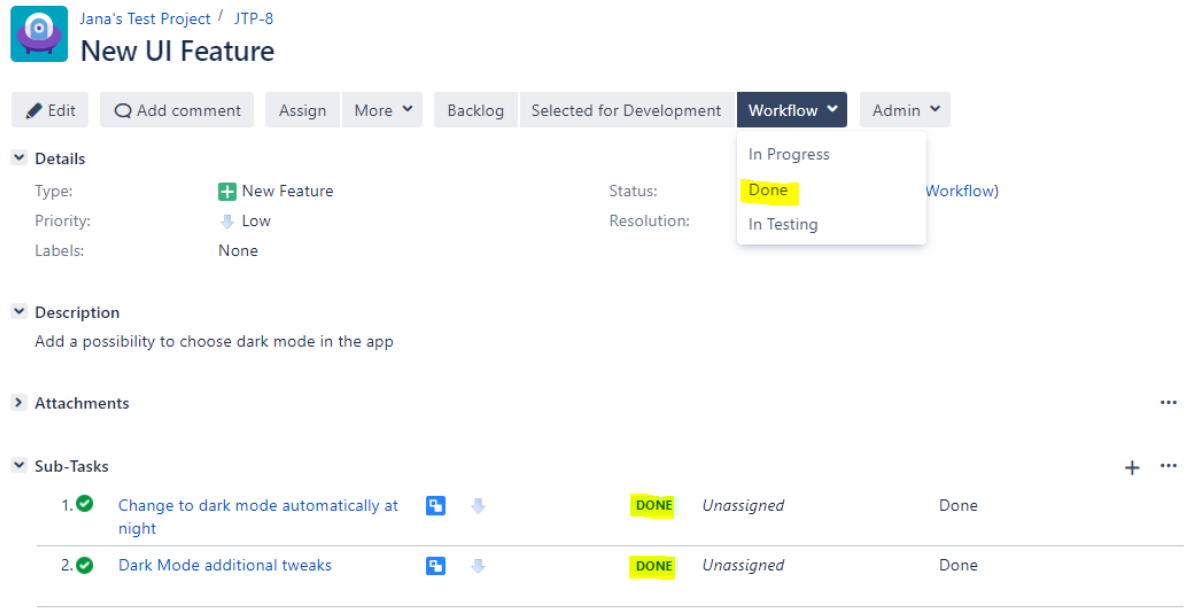


Figure 57. Done transition available once sub-tasks closed

The automation is set and working as expected. The purpose of this solution, as previously mentioned, does not directly affect performance, however, some users

prefer to have ‘cleaner’ UI with less unneeded buttons and elements. This case perfectly satisfies their needs.

3.3.6 Automatic closure of the parent issue when sub-tasks are completed

We have examined a specific scenario in which a parent issue cannot be closed until its child issues are incomplete. However, in this instance, our objective is to automatically close the parent task once all sub-tasks have been resolved. To accomplish this, we will utilize the Automation for Jira plugin to implement the solution.

In the project settings, open *Project Automation*. Click on *Create rule*. For ‘When’, choose *Issue transitioned* event and select *Done* in the *To status* box, as pictured in Figure 58. It is important to mention that the plugin exclusively displays the statuses that exist within the project.

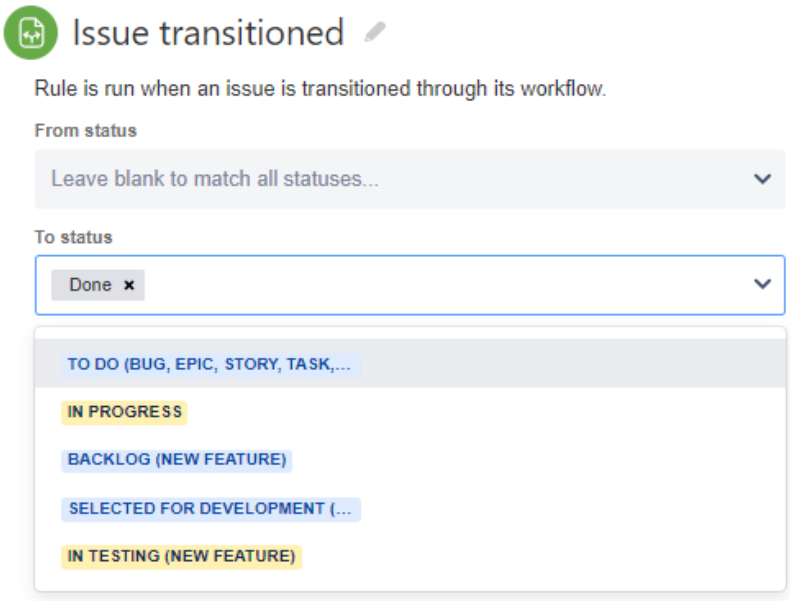


Figure 58. Issue transitioned event settings

Click *Save* and add *New Condition*. Choose *Issue fields* condition and choose issue type equals sub-task, as shown in Figure 59.

Figure 59. Issue fields condition settings for sub-task

Click *Save* and *Branch Rule*, then set *Parent* (Figure 60). Afterwards, click *Save.*

Figure 60. Branch rule / related issues settings

In the *For Parent* section of a rule, a condition must be added. As stated before, all sub-tasks must be closed, and then the parent task should automatically close as well. Figure 61 shows the condition to set up in For Parent section.

Figure 61. Issue fields condition settings for parent task

Click *Save* and add the last condition. Select *Related issues* condition, choose *Sub-tasks* and *All match specified JQL*. Write JQL query to include issues with status Done. Final result is demonstrated in Figure 62.

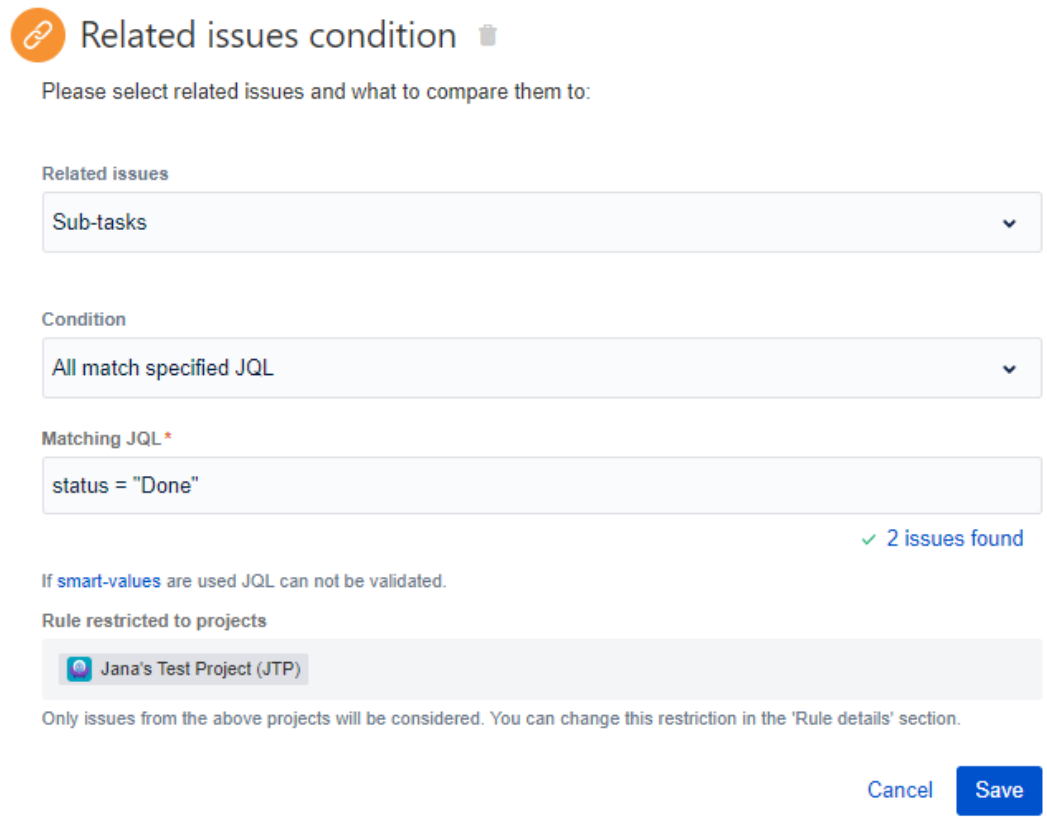


Figure 62. Related issues condition settings

Lastly, add the action which will be executed when all conditions are met. Click on *New Action* and choose *Transition Issue*. Select the first radio button to select the destination status, and set *Done* as a destination status. Click on *More options* to set additional option, such as adding a new comment once the rule executes correctly. Ensure that the settings are the same as in Figure 63, then click save.

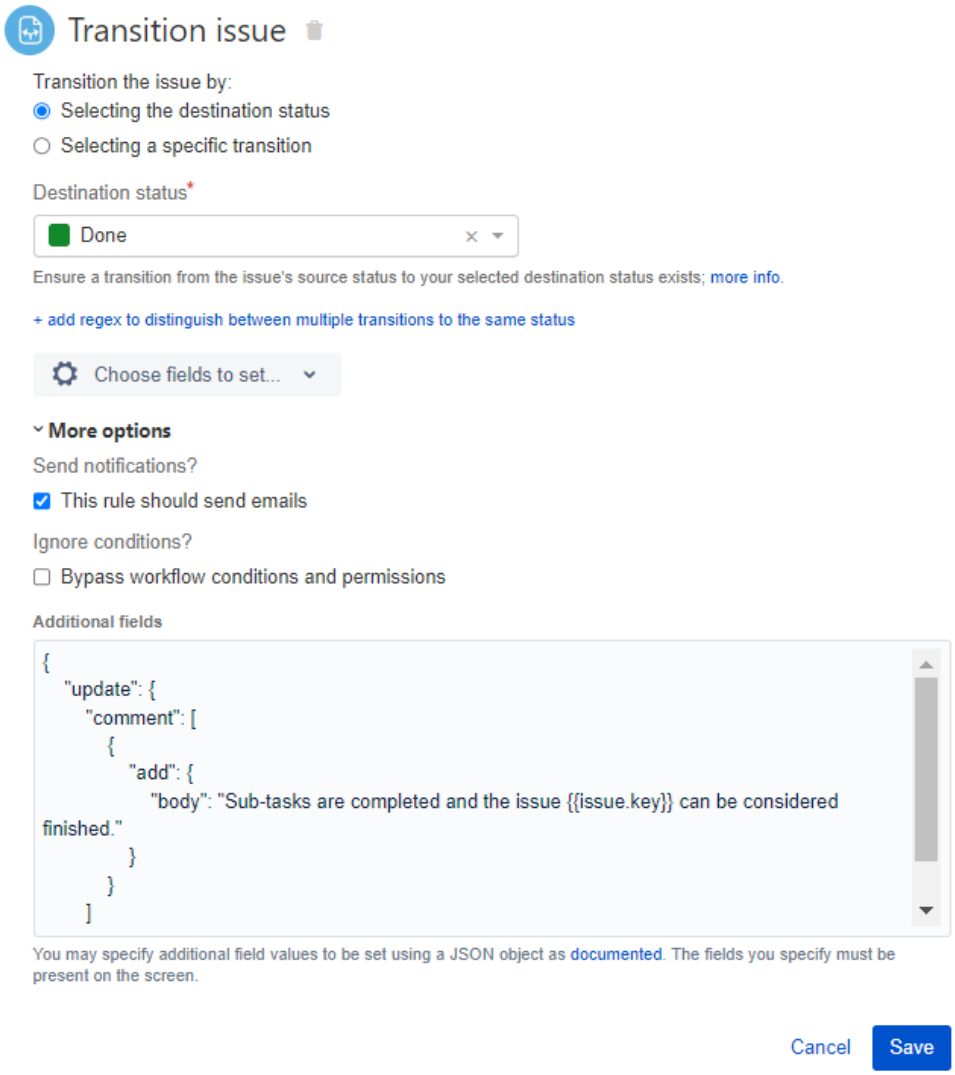


Figure 63. Transition issue settings

Additionally, Figure 64 includes the full JSON code which is not visible in the previous figure.

```
{
    "update": {
        "comment": [
            {
                "add": {
                    "body": "Sub-tasks are completed and the issue {{issue.key}} can be considered finished."
                }
            }
        ]
    }
}
```

Figure 64. Full JSON

Finally, provide a name for the automation and enable it as the last step. Figure 65 depicts both the previously added rule settings and the final step of adding the rule.

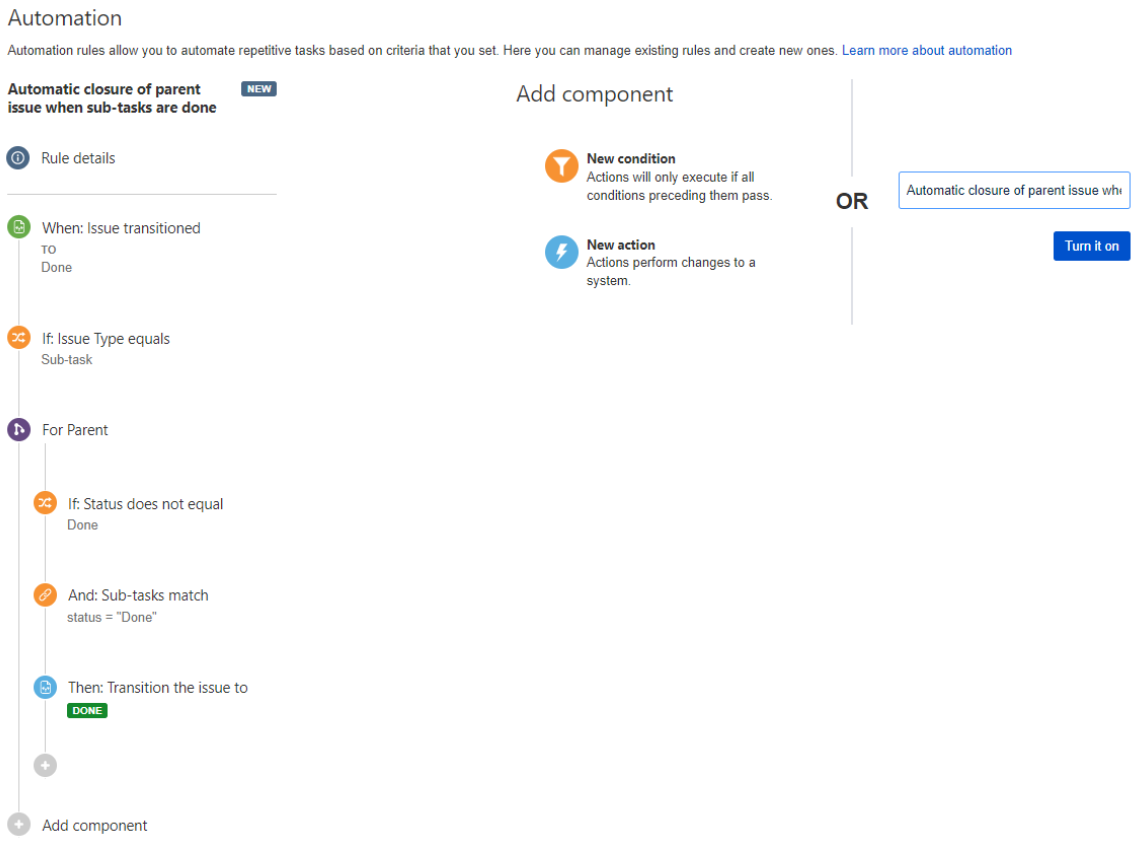


Figure 65. Final look of the automation rule before turning it on

To test the solution, create an issue with at least two sub-tasks. An example of a test solution is illustrated in Figure 66.

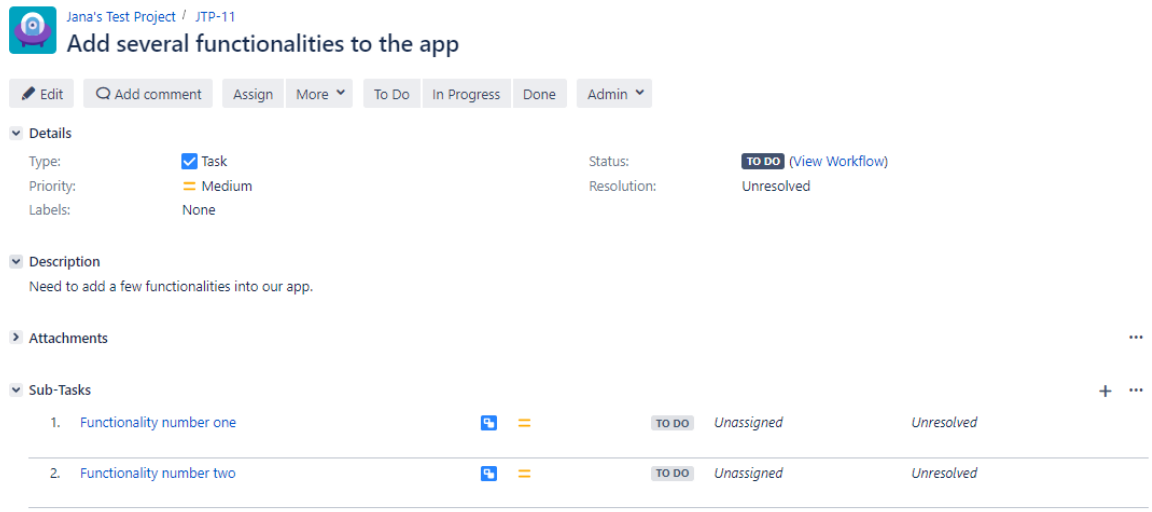


Figure 66. Issue with two sub-tasks, all incomplete

First, try to move the first sub-task to Done status, and examine the parent issue. No changes are done to the parent task (Figure 67).

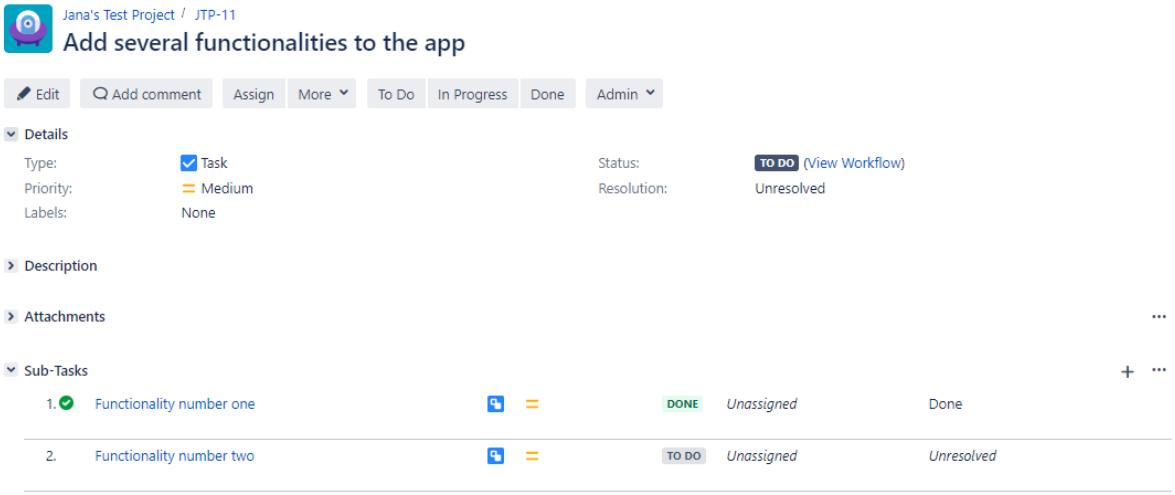


Figure 67. Issue with two sub-tasks, one sub-task complete

However, once the second sub-task moves to Done status, the parent issue also transitions to Done, as seen in Figure 68.

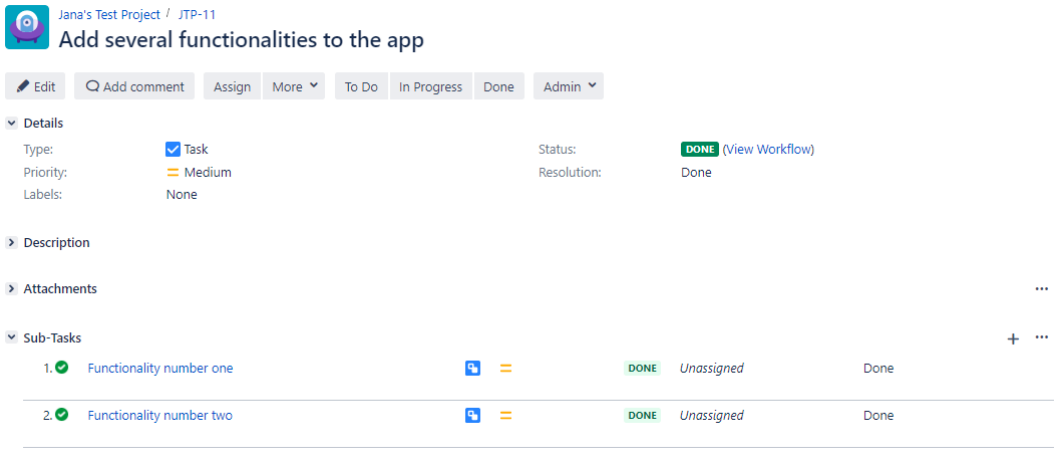


Figure 68. Issue with two sub-tasks, all complete and parent task closed automatically

Also, the comment was automatically added as well (Figure 69).

Figure 69. Comment automatically added to the parent issue

Furthermore, a similar automation can be given to Epic issues. Optionally, admins can set a ‘cascade’ method of reporting tasks. For example, an Epic called *Create a Game,* which has child issues in the form of Task issue type, named *Front-end developing* and *Back-end developing*. Each of them has sub-tasks, such as *Communicate with stakeholders*. To complete the Epic, teams must start from sub-tasks, gradually moving to bigger tasks.

3.3.7 Issue Matrix: detailed issue view under the Epic

To enhance accessibility and streamline users' routine work, administrators have the option to customize the Jira issue view screen using third-party plugins. For instance, plugin Issue Matrix can assist in such scenario. In our particular situation, we intend to implement a more comprehensive view of issues associated with the Epic, surpassing the limitations imposed by Jira's native functionality.

To implement the solution using Issue Matrix, a new field should be created. Go to *Issues* -> *Custom fields* -> *Add custom field*. Final page is illustrated in Figure 70.

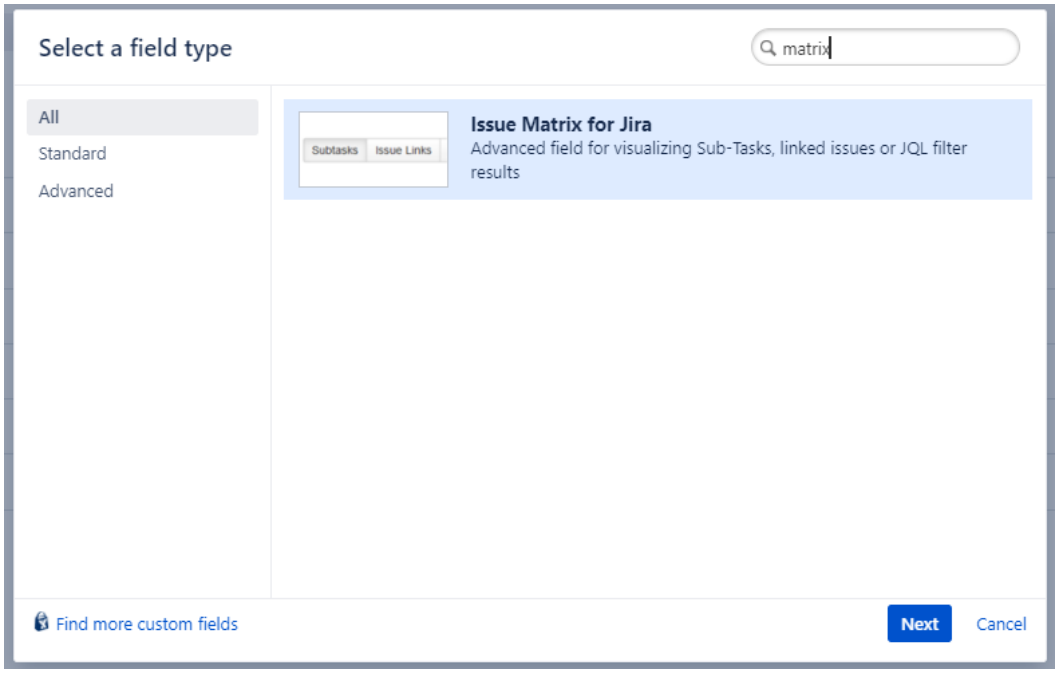


Figure 70. Creating an Issue Matrix field, stage 1

Name it *Issues in Epic* and optionally add description. It is important to configure context as detailed as possible. Select issue types where the field is expected to be seen (Epic, in this case), choose *Apply to issues in selected projects* and choose our project. Failing to specify the context of a field can result in future performance issues, such as affecting project and Jira instance indexing time. Final page is illustrated in Figure 71.

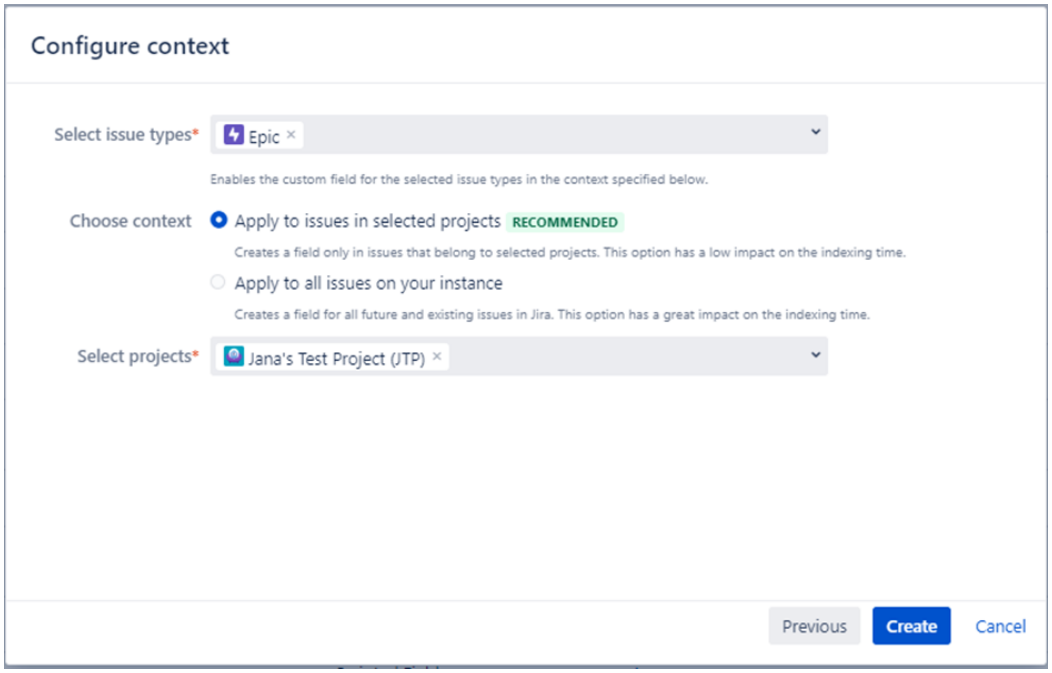


Figure 71. Creating an Issue Matrix field, stage 2

Click *Create* and associate with screens. Find the same screens used for Epic issue type in JTP and confirm to save the settings. Go back to *Custom fields* and search for the new field. Click *Configure*. Find *Edit Matrix Configuration* and click on it, as shown in Figure 72.

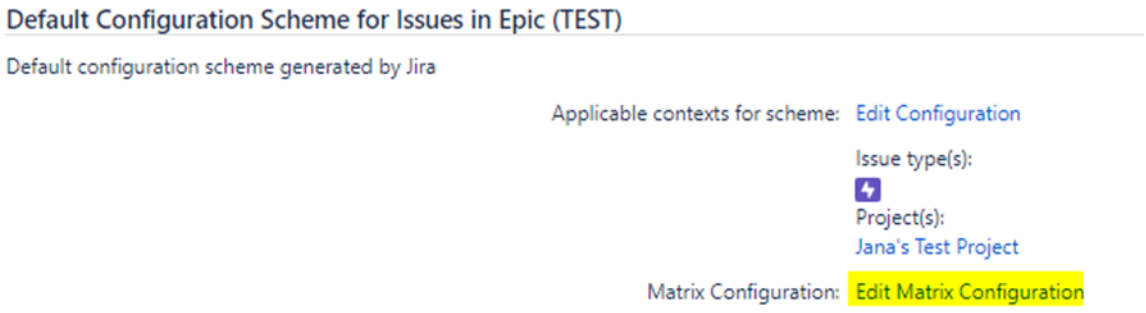


Figure 72. Field settings and configuration

Choose Epic mode for the field and move on to other settings. Choose all issue types to be visible under the Epic. Next, choose the fields to include in the list. For this case, I have chosen the most important fields: key, summary, priority, reporter, assignee and status. The plugin also allows us to specify the size (in pixels or as a percentage) of columns each of these fields corresponds to, however, when left unspecified, the tool will automatically adjust sizes perfectly. The final result of the first part of the page can be observed in Figure 73.

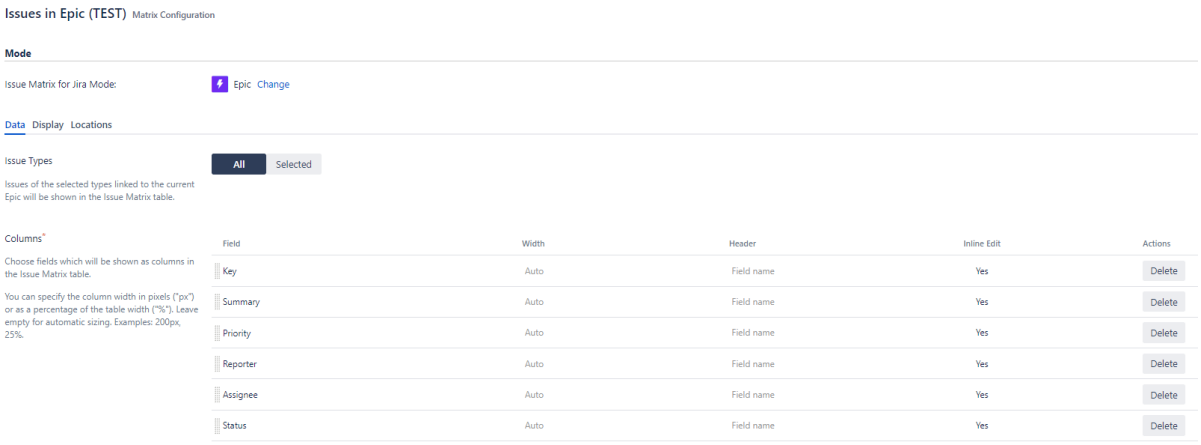


Figure 73. Issue Matrix field configuration, Data tab

Below the *Data* tab, the *Display* section can be seen. Click on *Show Table Headers* to use the field names mentioned before as a table header. Also tick *Show Empty Table* and select *All* to allow the table to include all issues. Figure 74 demonstrates the options selected for this case.

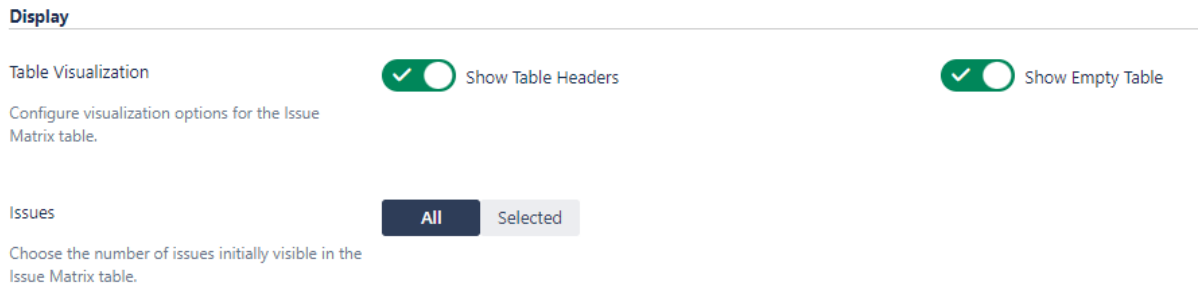


Figure 74. Issue Matrix field configuration, Display tab, part 1

Next, in *Jira Panels*, tick *Hide Issues in Epic* to remove the native Jira *Issues in Epic* tab from issue view screen, as seen in Figure 75.

Figure 75. Issue Matrix field configuration, Display tab, part 2

Lastly, the *Locations* section provides us with the flexibility to select our desired placement for the Issue Matrix table, based on our preferences. In Issue Detail View, select *Screen Fields Panel* (Figure 76) to position the table just below the *Details* and above the *Description* tab in Issue view screen. Leave other sections on default settings.

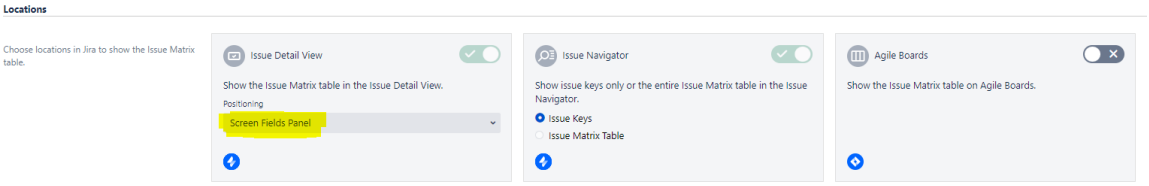


Figure 76. Issue Matrix field configuration, Locations tab

To test the solution we have implemented, create an Epic and several issues belonging to it. Figure 77 shows the issue view screen with newly created tab.

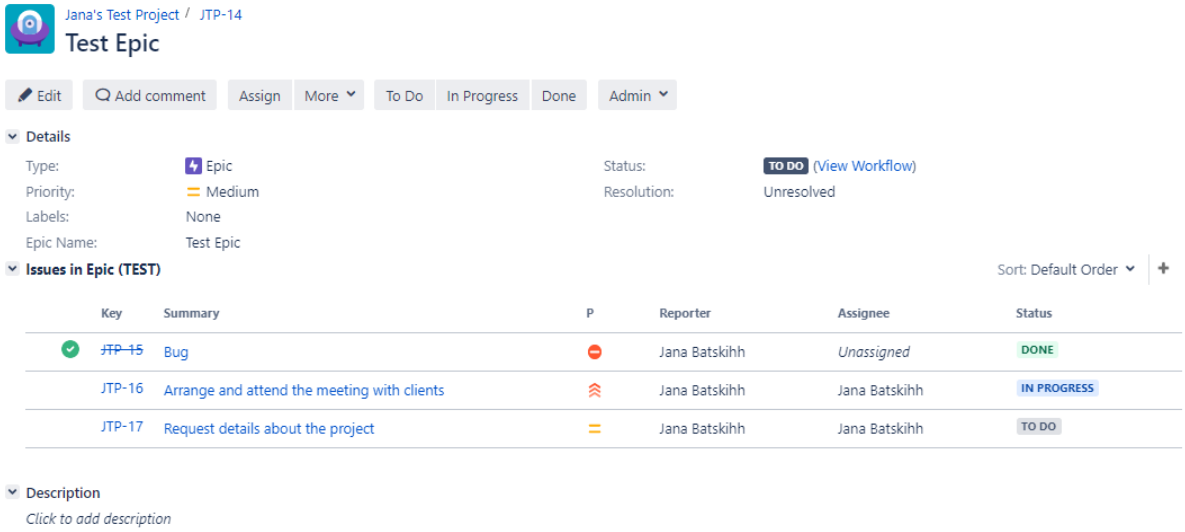


Figure 77. Issues in Epic tab visible in Epic issue view screen

This solution exemplifies how third-party plugins can significantly enhance the original functionality of Jira. The ability to customize existing elements is highly valued by users as it leads to a superior user experience and facilitates efficient work processes.

3.4 Review of results

We have explored various approaches to enhance users' daily routines in the use cases discussed in previous chapters. While Automation is not a fundamental principle of DevOps, it undeniably contributes to improved productivity and efficient utilization of working hours for users, leading to reduced errors and enhanced work performance.

Based on a small survey conducted within the company, users expressed that automating certain tasks had a positive impact on their work by expediting routine processes. Setting up an automation typically requires only 15-30 minutes, depending on the complexity, while each automated task saves approximately 3-5 minutes per issue, freeing up valuable time for users. On average, if a team generates 20 issues per day (roughly 440 issues per month), automation can potentially save up to 22-37 hours per month (equivalent to 444 hours per year!). This substantial number of saved hours enables teams to allocate their time to other critical tasks, for example, those which cannot be automated.

By streamlining workflows and freeing up significant amounts of time, automation empowers teams to focus on more important endeavors, thereby boosting overall productivity and contributing to the success of the organization.

4 CONCLUSION

The main objective of this study was to demonstrate how automating, simplifying, and optimizing users' daily tasks can facilitate adherence to the DevOps approach. In this context, the goal has been achieved. The cases presented in this study are applicable to a wide range of teams, regardless of their size or area of expertise. While there are additional methods for improving work using Jira that were not covered in this study, these examples provide an excellent starting point for many projects.

Several limitations should be acknowledged. Excessive automation can potentially degrade Jira's performance, so it is crucial to focus on implementing only the most essential automations and regularly update them after each Jira, Confluence, or integrated tool upgrade. Effective communication with stakeholders is also important, and conflicts or bugs may arise when integrating with certain plugins. While Jira and other Atlassian tools offer integration with numerous other tools and plugins, it is essential to recognize that not every integration will function seamlessly. In some cases, addressing issues with the intended automation can hinder the product development process.

Certainly, there are numerous ideas for further development. In addition to third-party plugins, Atlassian products, particularly Jira, offer customization options that can be implemented by clients themselves. It is important to consider that developing custom plugins internally requires company resources (both time and money) and may introduce bugs or conflicts with third-party plugins, as previously mentioned. Even in DevOps, maintaining balance is crucial, and teams must identify the most suitable approaches that align with their unique requirements, as no two teams are exactly alike. What works for one team may not necessarily work for another.

It is worth noting that DevOps is not solely reliant on Jira and Atlassian products. Other software, such as Microsoft's Azure DevOps, can also assist in implementing the DevOps approach. There are numerous other tools available, and each team or company should select the one that best suits their specific needs and budget. Ideally, the DevOps approach should be implemented across multiple software platforms, as focusing solely on one phase of the software lifecycle does not guarantee the optimal solution. However, there are no rigid rules prescribed for DevOps; it is a philosophy that can support teams in their tasks.

It is essential to remember that communication and collaboration are at the core of DevOps, with automation and task optimization being of secondary importance. Regardless of the tools used, effective communication remains the most critical aspect of DevOps. By establishing strong communication channels among team members, teams can thrive in their DevOps journey. Nevertheless, the software industry is constantly evolving, with new approaches to software development emerging and evolving needs. However, the significance of proper communication between individuals remains unchanged.

In conclusion, this thesis provides a valuable starting point for companies embarking on their DevOps journey and offers additional ideas for further development tailored to each company's specific needs. Throughout this experience, I have gained valuable insights and knowledge. As I continue my

work with the commissioner, I am enthusiastic about exploring and implementing more advanced solutions that will contribute to the success of our teams within the company.

REFERENCES

1 Adaptavist, 2023. *Post functions tutorial.* [Online]
Available at:
https://docs.adaptavist.com/sr4js/latest/features/workflows/workflow-functions-tutorial/post-functions-tutorial
[Accessed 26 May 2023].

2 Atlassian, 2020. *Atlassian survey 2020 - devops trends,* s.l.: s.n.

3 Atlassian, 2023. *Best practices on using the "resolution" field.* [Online]
Available at: https://confluence.atlassian.com/cloudkb/best-practices-on-using-the-resolution-field-968660796.html
[Accessed 26 May 2023].

4 Atlassian, 2023. *Bitbucket overview.* [Online]
Available at: https://www.atlassian.com/software/bitbucket/guides/getting-started/overview#a-brief-overview-of-bitbucket
[Accessed 8 May 2023].

5 Atlassian, 2023. *Code search and diff tool for svn, git and more.* [Online]
Available at: https://www.atlassian.com/software/fisheye
[Accessed 8 May 2023].

6 Atlassian, 2023. *Confluence - features.* [Online]
Available at: https://www.atlassian.com/software/confluence/features
[Accessed 8 May 2023].

7 Atlassian, 2023. *Crucible code review tool for git, svn, perforce and more.* [Online]
Available at: https://www.atlassian.com/software/crucible
[Accessed 8 May 2023].

8 Atlassian, 2023. *DevOps tools for each phase of the devops lifecycle.* [Online]
Available at: https://www.atlassian.com/devops/devops-tools
[Accessed 7 May 2023].

9 Atlassian, 2023. *Hosting options for jira software.* [Online]
Available at: https://www.atlassian.com/software/jira/guides/more/jira-hosting
[Accessed 8 May 2023].

10 Atlassian, 2023. *Open devops is the solution.* [Online]
Available at: https://www.atlassian.com/solutions/devops
[Accessed 8 May 2023].

11 Atlassian, 2023. *Opsgenie - features.* [Online]
Available at: https://www.atlassian.com/software/opsgenie/features
[Accessed 9 May 2023].

12 Atlassian, 2023. *What is agile?.* [Online]
Available at: https://www.atlassian.com/agile
[Accessed 8 May 2023].

13 Atlassian, 2023. *What is devops?.* [Online]
Available at: https://www.atlassian.com/devops
[Accessed 22 March 2023].

14 Buchanan, I., 2023. *CALMS framework.* [Online]
Available at: https://www.atlassian.com/devops/frameworks/calms-framework
[Accessed April 15 2023].

15 Buchanan, I., 2023. *Team topologies.* [Online]
Available at: https://www.atlassian.com/devops/frameworks/team-topologies
[Accessed 15 April 2023].

16 Churylov, M., 2023. *DevOps roles and responsibilities: building an effective team.* [Online]
Available at: https://www.mindk.com/blog/devops-roles/
[Accessed 31 March 2023].

17 Forsgren, N., 2019. *The 2019 accelerate state of devops: elite performance, productivity, and scaling.* [Online]
Available at: https://cloud.google.com/blog/products/devops-sre/the-2019-accelerate-state-of-devops-elite-performance-productivity-and-scaling
[Accessed 7 May 2023].

18 Hall, T., 2023. *4 key devops metrics to know.* [Online]
Available at: https://www.atlassian.com/devops/frameworks/devops-metrics
[Accessed 7 May 2023].

19 Hammond, J., 2011. *The relationship between dev-ops and continuous delivery: a conversation with Jez Humble of thoughtworks.* [Online]
Available at: http://blogs.forrester.com/jeffrey_hammond/11-09-09-

the_relationship_between_dev_ops_and_continuous_delivery_a_conversation_with_jez_humble_of_thought
[Accessed 8 May 2023].

20 Humble, J. & Farley, D., 2011. *Continuous delivery: reliable software releases through build, test, and deployment automation..* s.l.:Pearson Education Inc..

21 Kaufman, B., 2020. *Another way to gauge your devops performance according to dora.* [Online]
Available at: https://cloud.google.com/blog/products/devops-sre/another-way-to-gauge-your-devops-performance-according-to-dora
[Accessed 6 May 2023].

22 Krohn, R., 2023. *DevOps toolchain: key considerations.* [Online]
Available at: https://www.atlassian.com/devops/devops-tools/choose-devops-tools
[Accessed 5 May 2023].

23 Mohanan, R., 2022. *What is devops lifecycle? Definition, key components, and management best practices.* [Online]
Available at: https://www.spiceworks.com/tech/devops/articles/what-is-devops-lifecycle/
[Accessed 8 May 2023].

24 Openxcell, 2023. *DevOps - the complete guide for 2023.* [Online]
Available at: https://www.openxcell.com/devops/
[Accessed 28 May 2023].

25 Pennington, J., 2019. *The eight phases of a devops pipeline.* [Online]
Available at: https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba
[Accessed 6 May 2023].

26 Radigan, D., 2020. *JQL: the most flexible way to search jira.* [Online]
Available at: https://www.atlassian.com/blog/jira-software/jql-the-most-flexible-way-to-search-jira-14
[Accessed 8 May 2023].

27 Swartout, P., 2012. *Continuous delivery and devops: a quickstart guide..* s.l.:Packt Publishing.

28 Wikipedia, 2023. *Waterfall model.* [Online]
Available at: https://en.wikipedia.org/wiki/Waterfall_model
[Accessed 23 April 2023].

29 Zettler, K., 2023. *DevSecOps tools.* [Online]
Available at: https://www.atlassian.com/devops/devops-tools/devsecops-tools
[Accessed 1 May 2023].