"Quality Beyond Testing with DevOps: Jenkins to GitHub Actions Migration for Enhanced Process Optimization and Quality Assurance"

Sagar Aghera
saghera@netskope.com

Nikhil Yogesh Joshi
nikhilyogesh.joshi@fiserv.com

Amit Bhanushali
amit.bhanushali@mail.wvu.edu

Abstract

DevOps integrates and automates software development (Dev) with IT operations (Ops), complementing Agile SDLC by using automation tools to speed up software delivery. The choice of Continuous Integration/Continuous Deployment (CI/CD) tools is crucial for streamlining these processes. This paper presents our organization's migration from Jenkins to GitHub Actions for better scalability, security, and integration.

Previously, our organization faced challenges with Jenkins regarding scalability, security vulnerabilities, and complex third-party integrations. GitHub Actions provided a comprehensive platform for automating tasks within GitHub repositories. Our DevOps team automated processes like Pull Request checks, nightly builds, code coverage scans, and test executions. Integrations with JFrog Artifactory, Slack, Red Hat Ansible, and Atlassian JIRA facilitated a smooth transition.

GitHub Actions' secret management eliminated the need for additional tools like HashiCorp Vault and 1Password, enhancing security and simplifying secret storage. Its multi-language support, compatibility with major operating systems, and built-in secret management ensured a seamless migration. This paper discusses the challenges, benefits, and key considerations of the migration process, demonstrating how GitHub Actions improved our automation, workflow efficiency, and code quality assurance in a modern DevOps environment. After migrating to Github Actions, developer efficiency increased by 35%.

Index Terms—DevOps, Automation testing, CI, CD, Jenkins, Github Actions.

Biography

Sagar Aghera holds a master’s in computer science and a bachelor in electrical engineering from Florida Atlantic University, with over 10 years of experience in DevOps, CI/CD & test automation. He has worked at Motorola, Qualcomm, Gracenote, VMware, and currently excels as a Sr Staff Software Engineer at Netskope, focusing on testing frameworks, cloud infrastructure, and mentoring. His commitment to Testing and DevOps is evident throughout his career.

Nikhil Yogesh Joshi, a Senior Manager in Software Engineering based in Cumming, Georgia, boasts over 18 years of experience leading high-performing teams and delivering complex projects. His career spans multiple industries, demonstrating his expertise in automation, cloud migration, and strategic leadership.

Amit Bhanushali is a Manager, Software Quality Assurance at West Virginia University based in West Virginia, United States and have more than 20 years of experience in BFSI and Higher Ed Domains. He is a certified Scrum Master and holds a master’s degree in business data Analytics from West Virginia University in West Virginia, United States.

1 Introduction

1.1 Background

DevOps has become an increasingly important practice for organizations that need to keep up with the demands for rapid development and testing. In today's evolving software development landscape embracing DevOps practices has become crucial for companies looking to improve teamwork, between their development and operations teams. This shift in approach aims to boost how often code is deployed and enhance the quality of services. Contrary to popular belief, DevOps does not replace QA participation or oversight requirements. However, It will necessitate a change in how QA plans and does testing in a DevOps environment. DevOps quality assurance will require a considerable cultural shift away from traditional testing methods and more development, operational, and QA team collaboration. In an earlier study, the effectiveness of implementing DevOps practices in enhancing continuous delivery has been demonstrated (Sharma 2018).

DevOps transcends mere tool adoption; it represents a transformation that demands a reexamination of testing methods. This article delves into QAs evolving role in the age of DevOps focusing on the strategies, obstacles and future paths for quality assurance testing. Continuous testing emerges as vital in this scenario by weaving testing activities into every stage of development. As businesses strive to deliver top notch software the importance of QA approaches grows more evident. Continuous testing delivers continuous quality for any integrated CI/CD pipeline.

Traditionally the divide between Dev, QA & Ops teams – often termed as the "Wall of Confusion" – impeded communication and collaboration. Wall of Confusion here stands for Separation Of Duties (SOD). People often use this wall to get out undesirable tasks. To break this wall, DevOps practices were introduced. One can promptly resolve many QA issues in a DevOps environment by continuous improvement in communication and collaboration. The QA and development teams have traditionally operated in isolated groups, thus it's vital to initiate communication and ongoing feedback between them. DevOps seeks to break down this barrier by fostering an ethos towards software delivery. This approach stresses accountability, for service excellence with ongoing testing serving as a foundation. This provides an opportunity for QA to add an invaluable skill set in their repertoire.

1.2 Objectives

This article delves into the foundation of DevOps discussing its principles, advantages and tools particularly Jenkins and GitHub Actions which support CI/CD processes. Jenkins, an open-source automation server or platform has been widely employed to automate tasks but it does come with challenges such as maintenance overhead and security issues. Jenkins can be challenging to set up and maintain with outdated UI & requires regular upkeep & maintenance. It also demands careful configuration to ensure security, especially in public environments. On the other hand, GitHub Actions offers a streamlined approach by seamlessly integrating with GitHub repositories and providing robust support for automating workflows. GitHub Actions provides a user-friendly platform for developers & testers. GitHub Actions removes the necessity of managing servers, which leads to a decrease in operational expenses. GitHub Actions can run secluded jobs in secure environments, which come along with modern safety features like secrets management, access control, etc.

The shift from Jenkins to GitHub Actions marks an advancement in optimizing QA & DevOps procedures. This shift addresses issues linked to Jenkins like scalability problems and plugin dependencies while boosting efficiency and collaboration. The migration process involves planning, testing and validation to ensure a transition that upholds quality & DevOps standards.

Through case studies and practical applications this article showcases the advantages of embracing GitHub Actions for DevOps & QA purposes. These benefits include streamlined CI/CD workflows, quicker execution times and enhanced collaboration while making no compromises on security. The transition not only cuts down expenses but also elevates the overall reliability and security of the software delivery pipeline.

In summary, incorporating DevOps methodologies into quality assurance is crucial for upholding top notch standards in a changing software development landscape. With the increasing adoption of DevOps by companies QA approaches need to adapt to address emerging obstacles and make use of cutting-edge tools and practices. This document offers perspectives on the tactics and upcoming trends in QA during the DevOps era highlighting the significance of enhancements and teamwork.

2 Understanding Jenkins

2.1 Overview

As per Jenkins documentation, Jenkins is a self-contained, open-source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software (Jenkins Documentation). It helps automate the parts of software development related to building, testing, and deploying, facilitating DevOps practices by allowing developers to integrate changes into a shared repository frequently and obtain rapid feedback. According to the Jenkins Documentation, the pipeline features enable automation of software deployment processes effectively (Jenkins Documentation).

2.2 Strengths & weaknesses

Jenkins provides a high level of customization due to its open-source design, range of plugins and ability to integrate with different CI/CD systems. However users need to invest time and effort in configuring and maintaining it properly to deal with performance challenges, plugin compatibility issues and security concerns, which can be demanding and intricate tasks. Table.1 highlights a few strengths and weaknesses.

Strengths	Weaknesses
Open Source	**Complex Configuration**
Free to use and supported by a large community.	Can be complex to set up and configure, especially for beginners.
Flexible and Extensible	**Maintenance Overhead**
Highly customizable to fit various CI/CD workflows.	Requires regular maintenance and updates to ensure stability and security.
Active Community and Documentation	**User Interface**
Large, active community providing support and comprehensive documentation.	The UI can be seen as outdated and not as user-friendly as other CI/CD tools.
Cross-Platform	**Security**
Runs on various operating systems including Windows, macOS, and Linux.	Requires careful configuration to ensure security, especially in public-facing environments.

Table.1 Jenkins strengths vs weaknesses.

3 Introduction of GitHub Actions

As described in GitHub Actions documentation, GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment

pipeline. You can create workflows that build and test every pull request to your repository or deploy merged pull requests to production (GitHub Actions: Overview). It is an elegant solution that offers the convenience of creating and managing CI and CD workflows from GitHub, the largest code-hosting platform in the world (Heller 2021). There is a rich ecosystem of more than 20,000 3rd party actions, for use. Fig.1 highlights the key components of GitHub Actions. GitHub Actions Marketplace provides all the necessary integration points for most of the widely used tools & platforms such as slack, Azure, AWS, JIRA, Artifactory etc.

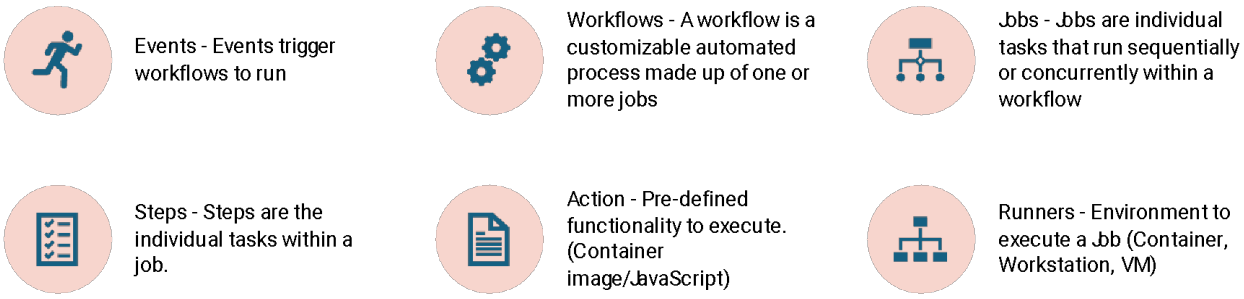

Fig.1 Key components of GitHub Actions.

4 Benefits of migrating to GitHub Actions

GitHub is by far the largest social coding platform, hosting the development history of millions of collaborative software repositories, and accommodating over 100 million users by January 2023 (100 Million Developers and Counting, 2023). GitHub Actions integrates closely with GitHub providing a safe and budget friendly CI/CD experience supported by a range of marketplace options. It boosts developer efficiency with an easy to use interface and adaptable workflows. Table.2 sheds light on key capabilities of GitHub Actions.

Capability	**Description**
Integration with GitHub	GitHub Actions is tightly coupled with the GitHub platform offering a unified workflow experience. This close integration enables Devs & QA to automate, personalize, and oversee their software development cycle within the GitHub repository.
Enhanced Developer Experience	GitHub Actions provides a user-friendly platform for developers & testers. By using YAML syntax to outline workflows, it becomes simple and approachable. Teams have the option to utilize actions from the GitHub Marketplace or design personalized actions tailored to their requirements.
Cost and Resource Efficiency	Using GitHub Actions removes the necessity of managing CI/CD servers, which leads to a decrease in operational expenses. It offers solutions that expand alongside the project, guaranteeing resource utilization.
Reliability & Security	GitHub Actions can run secluded jobs in secure environments, which come along with modern safety features like secrets management, access control, etc. It makes it possible to preserve sensitive data and information.
Ecosystem and	The GitHub Market provides a rich ecosystem of pre-built actions created by the

Capability	Description
Marketplace	community and industry leaders and curated by GitHub. This extensive library, comprising more than 20,000 actions, allows developers to easily integrate 3rd party tools into their workflows.

Table.2 GitHub Actions capabilities

5 Migration Process: Step-by-Step Guide

GitHub does provide Actions importer tool to facilitate migration from widely used tools like Azure DevOps, Bamboo, Jenkins, CircleCI and BitBucket but we didn't use this tool for our migration. As per Rautiainen, the migration still requires a lot of manual transformation and verification especially when custom scripts and nested and conditional steps are involved (Rautiainen, 2023). Migrate from Jenkins to GitHub Actions by analyzing the current setup, planning the transition, converting pipelines to GitHub workflows, testing in staging, and deploying to production while monitoring performance, as presented in Fig.2 and discussed in below sub-sections.

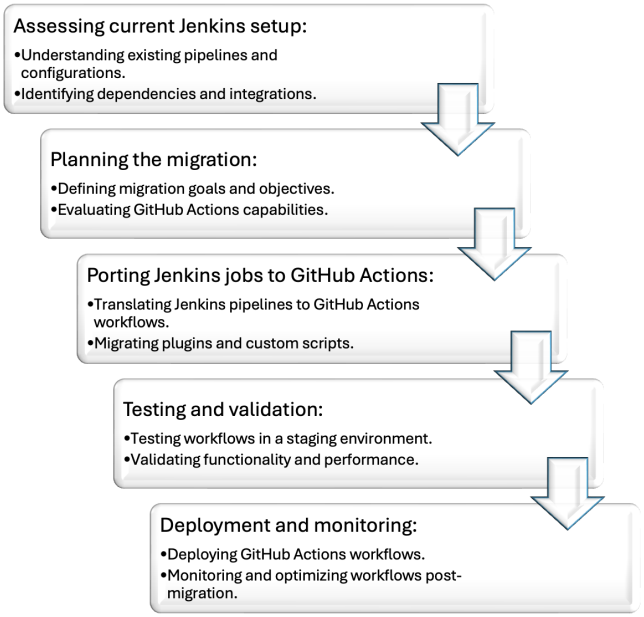

Fig.2 Migration process

For a smooth migration, the first step is to methodically analyze the current Jenkins setup which includes pipelines, freestyle jobs and configurations. Document it and additionally, identify any external plugins and custom scripts. These dependencies will need to be addressed during the migration. The second step is to plan the migration by identifying goals and objectives of migrating to the GitHub Actions. Mostly, it has to do with understanding of capabilities, features, and understanding of workflow syntax and available actions.

In the third step, the Jenkins job and pipeline are ported to GitHub action workflows. This will involve translating the Groovy based pipeline syntax to the YAML based GitHub Actions workflow syntax. Minor adjustment of the logic or functionality might be needed to align with GitHub Actions features. For 3rd

party plugins or custom scripts used in Jenkins, identify equivalent GitHub Actions built-in actions or 3rd party actions. One of the most frequently made mistakes is trying to migrate workflows in as-is status, the ability of GitHub Actions may allow us to simplify our workflows and reduce complexity, this often causes a new user to go down a rabbit-hole and waste time & energy. For example, in one of the case studies discussed in the next section, we were migrating a custom script to create tag and cut a release which was readily available as a built-in action.

At last, setting up the staging environment to test the GitHub Actions workflows meticulously. This will help in identifying any issues before we deploy to the production environment. Conduct comprehensive testing, including edge cases & failure scenarios to make sure workflows are robust enough. And if the performance gain was a criterion for migration, then make sure migrated workflows meet expectations. Once testing & validation is done, deploy the GitHub Actions to the production environment. Continue to monitor the performance & behavior of workflows and identify any areas of improvement if needed.

To further emphasize the key benefits of above migration practices, we compiled our results from 3 case studies which are described in the next section.

6 Case Studies

6.1 Case Study 1: Client builds pipeline

Netskope encountered difficulties, with their Jenkins build process, for Endpoint DLP client product. The challenges involved upkeep, intricate setup handling and prolonged build durations all of which impeded engineering productivity and caused release delays at times. The approach included transitioning from Jenkins to GitHub Actions to simplify and modernize the build pipeline. GitHub Actions was chosen for its compatibility with GitHub repositories, easy setup and robust automation features. Table.3 covers the pipeline steps involved when using Jenkins and after migrating to GitHub Actions. One notable advantage we had with migration is use of readily available Built-in action to tag the build and release which was done using custom script written & maintained by DevOps team. This resulted in per build time reduction from 50 mins in Jenkins to about 40 mins in GitHub Actions which is about 20% reduction in execution time. Additionally, we had more stable runs after migration, with failure rate reduction from 15% to 3%.

Pipeline Steps	Jenkins	GitHub Actions
Source code checkout	GitHub plugin	Built-in Action
Build product	Built-in shell plugin	Built-in Action
Run unit tests	Built-in shell plugin	Built-in Action
Upload artifacts	JFrog plugin	JFrog Action
Tag build & release	Custom script	Built-in Action
Send email	Email-ext post build action	Send Email Action
Slack notification	Slack Notification plugin	Slack API Action

Table.3 Pipeline steps – Jenkins vs GitHub Actions

6.2 Case Study 2: Pull request sanity

Given the Jenkins infrastructure and GitHub being in a separate network, Jenkins hosted in a local datacenter, webhook to GitHub from Jenkins server was not configurable. Pull request sanity check was a new addition to CI, given it was readily available with GitHub Actions. GitHub Actions is event driven and one of the supported events is Pull request. This made the creation of pull request sanity check quite trivial and many of the steps from client build pipeline in previous case study were reused. Fig.3 covers the steps involved in Pull request sanity check. This PR sanity check was more effective because it allowed us to isolate issues earlier, which would help in stopping issues entering the main branch. Additionally, code coverage check helped in following TDD (Test Driven Development) principles.

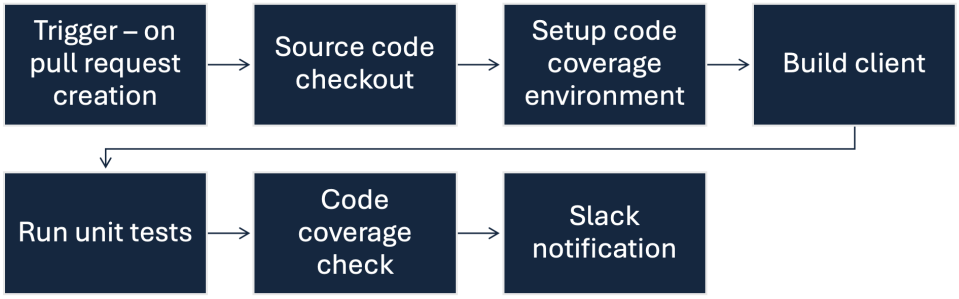

Fig.3 Pull request sanity check steps

6.3 Case Study 3: Test automation run(s)

Before migration, test automation runs were done using Jenkins freestyle projects. There was usage of 3rd party plugins – source code checkout, test report parser, custom email plugin, and slack notification plugin. Migrating to GitHub Actions involved matching with equivalent Actions and creating a workflow file with the steps calling those actions. One notable point here, the resultant workflow file contained less than 75 lines. Template of the workflow file in use is shown in Fig.4

The workflow had a trigger to run on schedule (every weekday). The workflow consisted on one job with below steps in order,

1. Automation source code checkout
2. Run python tests
3. Parse the junit report
4. Post the results to slack channel

```yaml

name: EPDLP Test Automation
on:
  schedule:
    - name: run smoke tests
      cron: "0 12 * * 1-5" # run every week day

jobs:
  run_smoke_tests:
    runs-on: [self-hosted, linux, x64, epdlp-qe] # Tests are run on self-hosted linux VM.
    steps:
      - name: Automation Checkout # Automation source code checkout
        uses: actions/checkout@v3.1.0

      - name: run PDVs # Running Python based test automation
        run: |
          pytest -m "smoke" --junitxml=smoke_tests.xml

      - name: Test Report # Parsing Junit report
        id: test-results
        uses: EnricoMi/publish-unit-test-result-action@v2
        with:
          action_fail: true
          files: |
            ./*.xml

      - name: Post to a Slack channel # Posting test outcome to slack channel
        uses: slackapi/slack-github-action@v1.24.0
        if: success() || failure()
        with:
          channel-id: <channel-id>
          slack-message: "Test outcome: ${{ fromJSON( steps.test-results.outputs.json ).conclusion }} \n
          Total tests: ${{ fromJSON( steps.test-results.outputs.json ).stats.tests }} \n
          ${{ fromJSON( steps.test-results.outputs.json ).stats.tests_succ }} passed \n
          ${{ fromJSON( steps.test-results.outputs.json ).stats.tests_fail }} failed"
        env:
          SLACK_BOT_TOKEN: ${{ secrets.SLACK_TOKEN }}

```

Fig.4 Template of workflow file for test automation run(s)

6.4 Results and outcome

According to Smith, GitHub Actions significantly enhance the efficiency of workflow automation in development environments (Smith, 2020). The tight integration of GitHub Actions with GitHub streamlines the entire workflow process, and hence eliminating the need to manage separate systems for version control and CI/CD. By eliminating the costs associated with infrastructure maintenance, scaling, and support, about 15% reduction in operational costs was achieved after migrating from Jenkins to GitHub Actions. Developer productivity increased by about 10% due to tight integration and less context switching. After migrating to GitHub Actions, developers & QA can manage their code repositories, monitor CI/CD & automation runs all within the same platform and hence eliminate context-switching between different tools & interfaces. Additionally, code formatting/linting, and dependency management were also integrated using GitHub Actions built-in support, which helped in automating these repetitive tasks thereby helping the team focus more on writing quality code. These outcomes were observed from the three case studies conducted and mentioned in previous sections.

GitHub Actions provided highly optimized & scalable runners in form virtual machines & containers for executing workflows. They were efficient & responsive which allowed faster execution of CI/CD and automation runs when compared to self-hosted infrastructure. Additionally, the declarative YAML used to write workflows was much simpler and more readable. This made it easier to setup and maintain CI/CD pipelines, and thereby reducing the time spent on configuration and maintenance tasks. This led to about 10% less time spent on the CI/CD and automation after GitHub Actions migration.

After migration, we did analysis of the time spent with GitHub Actions when compared to Jenkins before migration. For every 100 hours spent with Jenkins, we were spending 65 hours with GitHub Actions. Thus we can conclude that we achieved about 35% efficiency after migration.

7 Best Practices for Migration of GitHub Actions

7.1 Incremental migration

Incremental migration involves the transitioning of the workflows progressively. Rule of thumb here is to start with a simpler and low-risk task. In this way, we minimize risks & disruptions by allowing exhaustive testing and configuration for each flow before moving to the next one. Additionally, it enables continuous learning and improvement, making the migration process more manageable and efficient.

7.2 Security Best Practices

Use GitHub Actions secrets management to store API keys, and passwords, ensuring they are not hardcoded in workflows. Monitor and assess audit logs and workflows logs to monitor for any unusual activities and any exposed secrets. Using the principle of least privilege, regularly review permissions for actions and workflows. Keep the actions up to date to avoid any vulnerabilities, and possibly use signed actions wherever possible. Another way of securing workflows is pin 3rd party actions. To ensure that the source code of the version you're using hasn't been altered you must pin the action to the full-length commit SHA. The value of commit SHA is unique for each commit and is immutable. This ensures that your GitHub Actions workflows will always use the same release of the Action.

8 Common Challenges and Solutions

8.1 Technical challenges

Migrating to GitHub Actions offers exciting opportunities to modernize automation and CI/CD pipelines, though it may pose a few technical challenges. Translating existing automation & CI/CD configurations to the new syntax does involve careful planning and validation. Given the goal is to maintain or enhance security posture while migrating, configuring secrets and permissions securely is essential. For someone not familiar with GitHub UI and system and/or declarative YAML syntax, it may be a bit of a learning curve. Thorough testing and documentation are necessary to ensure functionality and team readiness. Tackling these challenges systematically can lead to a successful migration of GitHub Actions.

8.2 Organizational Challenges

One of the organization challenges faced in managing change resistance by team or team members accustomed with existing tools. Additionally, coordinating across multiple teams and aligning on newer workflows and best practices can get complex. Allocate sufficient resources and time for smooth transition to avoid overburdening and maintaining productivity during migration. Organize Tech-talk(s), office hours for collaboration and knowledge sharing. Comprehensive training and upskilling across board is required to ease the migration process and avoid any disruptions or delays.

9 Conclusion

9.1 Cultural Contribution

Over a period, which started with the migration process to its completion, the change/shift in the culture of the team was starkly visible. From being multiple teams, and to catering multiple profiles [Dev vs QA vs DevOps Engineer], we now noticed a uniformity in approach towards pipelines. The synergy that now drives the overall process is geared towards keeping the pipeline a standard tool, which is the frontline candidate for shipping a product.

The team was missing a mindset which was sustainable to maintain a DevOps lifestyle, with the advent of GitHub Actions, we notice that the pipeline is now a first-class citizen, with SecOps as a front-end driver. The security enhancements that we were able to introduce and provide have been a boon for the team.

9.2 Positive Impacts

As mentioned before, the change in mindset of the team has brought in multiple positives, e.g. with the advent of GitHub Actions, the infrastructure management, and the constant maintenance and scalability were now a worry of the past. The extra time can be utilized in improving the product and implementing new ideas, the QA can focus on product automation and the DevOps can enable the team to sustain a quality pipeline, which is not compromised by vulnerable artifacts. The standardization is a boon in terms of Audits and Quality Checks which come as a part and parcel of Enterprise Software teams.

The Return of Investment is an often-discussed relevant topic, the total investment in terms of time and cost, and the total man-hours saved by implementing GitHub Actions is a net positive number, which for privacy reasons cannot be discussed, overall, the time saved is 35% more than when using the older technology stack.

9.3 Quality beyond Testing

The biggest takeaway from this exercise was the improvement seen in Automation. Since the platform was common and the exposure was greater it allowed the Automation code to be validated by Senior Developers, giving the benefit of added reviews. It also brought in a common platform to secure the libraries and ensure that the latest and often the greatest were parts of automation frameworks. This led to better quality and stability in framework and in turn the automation caught more bugs than before.

9.4 Future Trends

The integration with CoPilot, is something to which the team is looking forward to, the hope that AI will bring an efficiency jump and that the pipelines will be able to detect a CrowdStrike like failure well in advance.

10 References

Mayank Sharma and Indrajit Rajput, "Enhancing Continuous Delivery Using DevOps Practices," *Journal of Systems and Software* 136 (2018): 110-119.

Jenkins Documentation. "User Handbook Overview." Last modified 2024. Accessed July 14, 2024. https://www.jenkins.io/doc/book/getting-started/.

Jenkins Documentation. "Pipeline Documentation." Last modified 2024. Accessed July 14, 2024. https://www.jenkins.io/doc/book/pipeline/.

Thomas Dohmke, "100 Million Developers and Counting," The GitHub Blog, January 25, 2023, https://github.blog/2023-01-25-100-million-developers-and-counting/.

GitHub. "GitHub Actions: Overview." GitHub Docs. Accessed July 14, 2024. https://docs.github.com/en/actions.

Smith, John. "Automating Workflows with GitHub Actions." *DevOps Journal* 12, no. 3 (2020): 45-58.

Heller, Priscila. *Automating Workflows with GitHub Actions: Automate software development workflows and seamlessly deploy your applications using GitHub Actions*. Packt Publishing Ltd, 2021.

Rautiainen, Olli. "GitHub Enterprise and Migration of CI/CD Pipelines from Azure DevOps to GitHub." (2023).