Addressing CI/CD with Automated Testing in Your Software Estimate

Abstract

Agile development and DevOps/DevSecOps practices are becoming increasingly common in software development projects in commercial, government and government contractor organizations. Four practices arising are Continuous Integration, Continuous Delivery, Continuous Deployment, and Continuous Monitoring. Automated Testing, an important capability in this continuum of DevOps practices, is clearly an investment for an organization, but one that promises to lead to productivity and quality improvements. This paper presents the general concepts of automated testing focused on the practices highlighted above. Cost impacts and success metrics will be demonstrated and discussed.

1. Introduction

The State of DevOps Report for 2021 [1] found that 90% of respondents with highly evolved DevOps practices report that most repetitive tasks in their process have been automated and agree that automation improves the quality of their work. In a report published by Atlassian [3] on DevOps Trends reports that over half of the respondents reported that their organizations had dedicated DevOps teams, though investigation of recent State of DevOps Reports [1],[2] indicates that many organizations that claim to have dedicated DevOps teams may not be fully addressing all of the practices necessary to achieve organizational success with DevOps

The DevOps approach merges the skills of Development (Dev) with those of Operations (Ops) to optimize software delivery efficiency and customer satisfaction. One important premise of the DevOps approach is to create a culture where an organization embraces the value added to the software delivery process through automation, collaboration, continuous process improvement, rapid delivery and better resource and cost management. Adaption of processes such as Continuous Integration (CI), Continuous Delivery (CD), Continuous Deployment (unfortunately also often abbreviate CD) and Continuous Monitoring (CM) are important success factors for organizations pursuing DevOps successes. CI/CD bridges the gap between development and operation activities by enforcing automation in building, testing, delivery and deployment of applications. Automation of tests and delivery ensures that well tested and verified features are delivered regularly and promptly to end users.

This paper is focused on the above-mentioned practices (CI/CD/CD/CM) and what the potential impacts they have on productivity, product quality and cost. The rest of this paper is structured as follows, In section 2, definitions are provided for those unfamiliar with agile methodologies and DevOps/DevSecOps. Section 3 delves into the CI/CD (Delivery in this case) Pipeline – how it is implemented, how it is used and the benefits and challenges of creating, developing and maintaining a CI/CD pipeline. Section 4 covers Continuous Monitoring and presents a discussion of metrics that help to identify high performing DevOps teams and their impacts on software development and delivery. Section 5 discusses the cost/effort estimation considerations when a project includes CI/CD in their software development and delivery processes. The final section provides conclusions.

2. Definitions

DevOps/DevSecOps

The DevOps approach creates an environment that emphasizes collaboration and communication between Development and Operations, ensuring that these activities occur in tandem rather than serially. This requires that developers must take ownership of application's reliability and performance while business and project leaders must support and foster this shift in practices. Recognizing that software development and deployment is different than hardware development and deployment, the software industry, over the last 20 years, has increasingly adopted methodologies that support frequent releases of small software features and defect corrections. DevOps is a combination of cultural philosophies, practices and tools which, when properly and thoughtfully implemented, increase an organization's ability to deliver quality applications and services with high velocity. The DevSecOps definition is the same as DevOps but with an addition of the idea that security is "shifted to the left." So instead of waiting till the end of a release to think about security, security engineers are involved from the concept throughout the life of the application. DevOps/DevSecOps creates an environment where applications evolve and improve at a faster pace than applications developed using more traditional software development and infrastructure management processes.

Continuous Integration (CI)

Continuous Integration is emerging as one of the biggest success stories in automated software engineering [4]. CI is a practice that involves automatically building and testing code changes as they are committed to a code repository. CI focuses on integrating work from individual developers into a main repository multiple times a day to catch integration bugs early and encourage collaboration among the team. The goal of CI is to minimize the cost of integration by making it an early and often operation. Errors are detected early in the development process to prevent them from being released to production or at least to minimize the negative impact if they are released to production. CI relies on robust test suites and an automated system to run those tests. When a developer commits code to the main repository of the version control system, an automated process will be kicked off to create a new build of the application and run a suite of automated tests to ensure that no new changes have disrupted the integrity of the application.

Some of the benefits of CI include:

- Early detection of bugs improves quality and velocity. It is well understood in the software industry that the earlier bugs are detected, the cheaper they are to fix. When builds are created every time code is committed, or several times a day, the developer has total recall of what changes were made and is able to quickly identify and correct problems
- Feedback from developers, customers and stakeholders can be near real-time.
- Improved collaboration creates visibility within DevOps teams and the organization. With a shared platform for testing and building code, developers have improved collaboration opportunities.

Continuous Delivery (CD)

Continuous Delivery is an extension of CI that focuses on automating the software delivery process so that teams can quickly, easily and confidently deploy their applications to production at any time. The core tenet of CD is keeping the codebase in a state where it can be shipped to production at any time. Working this way teams can quicken the tempo of production changes, from infrequent, big and risky deployments to deployments that are frequent, small and safe [5]. CI is a prerequisite for CD because developers need to believe that the state of the current build is good enough to release. As the build moves through the

pipeline, it is delivered to environments that mirror the production environment as closely as possible. CD automates the steps that occur between code commit (or check in) and the decision as to whether it is of adequate quality to be released to the production environment. The steps that get to this build are automated; the final decision about when to release is still left to a human gatekeeper(s).

Some of the benefits of CD include:

- Faster time to market because developers can make frequent releases of new features and defect corrections, reducing time to market and improving customer delight.
- Reduced Deployment Failures as CD automates the process of delivery and performs the test suites to ensure that the code is functional.
- Improved Visibility and Collaboration since CD creates an environment for improved collaboration between development, operations and IT which improves quality and decreases friction.
- Improved productivity and efficiency because CD can result in significant time savings for developers, testers, and operations engineers by automating repeatable tasks. This frees developers up to spend more time doing creative tasks.

Continuous Deployment (also CD)

Continuous Deployment is an extension of Continuous Delivery that automatically deploys each build that passes the test cycle directly into the production environment, instead of waiting for a human gatekeeper to decide what and when to deploy. Continuous Deployments create an environment that encourages small changes with limited scope, where new features and bug fixes can be deployed to the customer community quickly and the overall deployment process is streamlined. In continuous delivery a release candidate is sent to human stakeholders where it is approved and then deployed. In continuous deployment, the build automatically deploys as soon as it passes its test suites. [7]

Some benefits of CD include:

- Faster Release Cycle because automated deployment processes enable faster releases and eliminate human gatekeeper.
- Continuous Improvement as CD creates a culture that encourages continuous improvement since developers can rapidly test and deploy new features and fixes.
- Improved Productivity because CD creates an environment that allows developers to focus on coding rather than deployment, this allows developers to spend more time on new development and less time on manual tasks.

While Continuous Deployment is an interesting topic, Continuous Delivery is the more reasonable option for the focus of this papre. For the remainder of this paper CD will indicate Continuous Delivery rather than Continuous Deployment.

Continuous Monitoring (CM)

With DevOps and other agile methodologies, an important practice is the frequent delivery of new content to end users. Automation is key to making this happen, and successful automation depends on Continuous Integration, Continuous Testing, Continuous Delivery and Continuous Monitoring. CM involves the continuous collection, analysis and reporting of key performance metrics to identify issues and optimize the performance of the software development life cycle. The primary goals of CM are:

- Enhance transparency and visibility of IT and network operations, especially focused on discovery and addressing potential security breaches
- Monitor Software Operations, especially focused on identifying and solving performance issues.
- Provide rapid feedback to the Development, Operations and Quality teams to optimize the performance of applications and ensure a great customer experience.

Some benefits of CM include:

- Improved security through automation
- Early detection of performance errors improves response time.
- Reduced downtime as issues are detected and addressed quickly.
- Better business performance resulting in improved user experience.
- Increased clarity and collaboration throughout the team.

3. Continuous Integration/Continuous Delivery (CI/CD) Pipeline

A continuous integration and delivery pipeline (CI/CD Pipeline) is an organized and automated suite of tests that is used to ensure all software versions are up-to date, compatible and ready to use [6]. According to a recent report from Puppet, organizations utilizing CI/CD practices deploy 46 times more frequently and spend up to 44% more time on creative work [8]. Pipelines are usually driven by tests, services, outcomes and occasionally people. A good CI/CD pipeline should deliver speed, consistency, tight version control, extensive automation, integrated feedback loops and security best practices throughout the pipeline.

The core components of any CI/CD pipeline are:

- Build scripts which are sets of instructions that are used to automate the tasks necessary to build a release of the application.
- Infrastructure elements such as development environments, platform(s), hosting services, or virtual machines
- Test elements which allow developers to create scripts to automate testing.
- Release elements which allow for the automation of the release process.
- Validation elements which are tools that are used to automate testing as part of the release process.

In *Continuous Delivery in the Wild [5]* research was conducted across a wide variety of organizations with rapid software delivery tempos. The path to production across the organizations was quite similar:

- Engineers implement features or other changes.
- New features are reviewed and merged into the master codebase.
- Build is created and automated tests validate the changes.
- Change is automatically delivered to a shared integration environment.
- Some exploratory testing is accomplished when necessary.
- Change is deployed to a production or production like environment.
- There is potentially a controlled rollout to the end users.

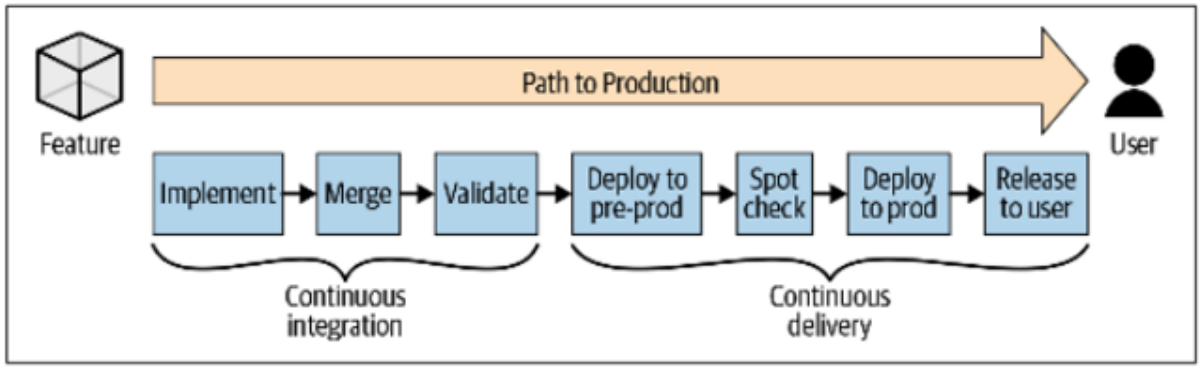

Figure 1 CI/CD Pipeline (Source [5])

Figure 1 shows the CI/CD Pipeline; the first three steps demonstrate the process of CI, while the rest of the process is considered CD.

The benefits of a CI/CD Pipeline include:

- Efficient software development created through smaller iterations and more efficient and automated testing. Limited scope makes it easier to find and fix bugs and feedback on newly implemented features occurs quickly.
- Products developed with CI/CD can reach the market faster and achieve more success with customers .
- The rapid cycles enable developers to experiment with coding styles and potential solutions with far less risk than a more traditional deployment. This approach offers an opportunity for Freedom to Fail as part of the developer experience.
- The constant flow of a CI/CD pipeline makes it easier to identify and fix bugs faster improving, the software maintenance activities.
- Logs generated by a CI/CD pipeline at every level of the software development process create near instant feedback and visibility into what's going well in the process and what is not.
- A well implement CI/CD pipeline can drastically improve code quality which in turn leads to cost effectiveness and overall ROI (Return on Investment) improvements.

There are also several challenges with CI/CD Pipelines which include:

- CI/CD relies on serious dedication to automation. Tool sets need to be established to create automation necessary to build, test and deploy. This calls for a serious investment up front, not only for the tools but for the time and effort required for human resources to learn the tools and create initial scripts and frameworks
- A CI/CD Pipeline requires planning and discipline with both the development teams and the project or program managers. Acceptance of CI/CD Pipeline paradigm requires buy-in, dedication, adherence to standards, and support throughout the organization.
- CI/CD is a culture change for an organization familiar with more traditional software development environments. An organization can invest heavily in tools and automation but without good communication and collaboration they will not be a successful DevOps Team; just a team with

lots of tools and automation. Interaction between developers, stakeholders and project management facilitates the rapid, efficient process that makes CI/CD so powerful.

4. Key Software Delivery Performance Metrics

According to Gartner, 87% of business leaders believe that digitization is an important priority for their organizations. DevOps transformation delivering this type of digitization requires a dedicated, continuous learning and improvement process to successfully reach maturity. Continuous monitoring should be an important part of the DevOps lifecycle, but it is frequently overlooked [9].

DevOPs Research and Assessment (DORA) has conducted an extensive academic study exploring DevOps principals and applications since 2014. They employ data driven insights to study and promote DevOps best practices by examining the differences with respect to these practices in high performing organizations and low performing organizations. Between 2014 and 2020, they studied DevOps teams and the progression that DevOps has had in the software industry and each year they have published a State of DevOps Report with their findings and recommendations. ([1] & [2] reference the 2021 and 2022 reports).

One of the important results of this long-term study were the publication of four key metrics that discriminate high performing DevOps teams from low performing ones. Each year the research gives guidelines as to how to measure whether a team is high, medium or low performing based on how they measure up to these four metrics. The four metrics they identified in 2020 were:

- **Deployment Frequency**

  Deployment Frequency is considered a measure of throughput. It measures the number of deployments in a given time frame. The higher the number the better because value is delivered more often. Deployment frequency is measured by using a pipeline tool to determine the total number of builds delivered and the number of successful builds delivered. Simply divide successful builds by total builds and multiply by 100 to get the Deployment Frequency. According to the 2022 State of DevOps Report from DORA

  - High performing teams deploy on demand (multiple deploys per day)
  - Medium performing teams deploy between once a week and once a month.
  - Low performing teams deploy between once a week and once every 6 months.

- **Lead Time for Changes (also called Cycle Time)**

  Lead Time for Changes is considered a measure of throughput. It measures the time to implement, test and deliver code for a feature (from first commit to deployment), including development, testing and delivery. According to Marc Anderson “cycle time compression may be the most underestimated force in determining winners and losers in tech” [10] The shorter the Lead Time the Better. To calculate this metric one needs to identify start of work and its finish. Improved automation and test integration will help an organization improve their performance on this metric. According to the 2022 State of DevOps Report from DORA:

- High performing teams have a mean time between one day and one week
- Medium performing teams have a mean time between one week and one month
- Low performing teams have a mean time between one month and six months

- **Mean Time to Recovery**

  Mean Time to Recovery (MTTR) is considered a measure of stability. This measures the time it takes an organization to restore a system when there has been a production failure. This includes both detection and response of a failure. Captures of the defect could be health checks, critical path end-to-end testing, testing in production, customer reporting and support reporting. MTTR can be calculated by tracking average time between a defect report and a fix being deployed to production. Continuous Monitoring tools can be configured to enable this tracking, providing alerts about potential problems. The shorter the MTTR the better. According to the 2022 State of DevOps Report from DORA:

  - High performing teams average less than one day.
  - Medium performing teams average between one day and one week
  - Low performing teams average between one week and one month

- **Change Failure Rates**

  Change Failure Rate is considered a measure of stability. This measures the number of code changes that result in any sort of production failure. In other words, this is the percentage of deployments that caused a failure in production. This can be calculated by dividing the number of deployment failures by the number of total deployments. Change Failure Rate can be decreased through robust monitoring and through strategies such as delivering in small increments and increased test automation. The lower the Change Failure Rate the better. According to the 2022 State of DevOps Report from DORA:

  - High performing teams have a rate between 0-15%
  - Medium performing teams have a rate between 16-30%
  - Low performing teams are between 46-60%

According to Nicole Forsgren [10] “High performers, year over year, are twice as likely to exceed their organizational goals of productivity, profitability and market share. We added additional measures like non-commercial goals and the crazy thing is we find this two times multiplier continuous to apply. High performing organizations are twice as likely to exceed their non-commercial goals. Indeed companies which have done well under these DevOps metrics have 50% higher market cap growth over 3 years”

5. Cost Considerations for DevOps and CI/CD with Automated Testing

There are definitely potential benefits of employing a CI/CD Pipeline, many of which have been discussed above. However, CI/CD is not inexpensive, and teams should consider whether the costs of deploying

and/or employing a CI/CD Pipeline makes sense for the type of project being estimated. There are many types of projects where the benefits make the investment a smart move. There are however, some situations where the decision to use the CI/CD Pipeline may mean the project costs more than if other more traditional software delivery processes are employed. Some project factors that have the potential to make this true:

- Projects where software deliveries are regularly customized for individual customers.
- Projects where there are compliance and/or security requirements that make automated testing complex or not acceptable.
- Projects that integrate many third-party solutions.
- Project where Service Level Agreements may require customized pipelines.
- Projects with a great deal of algorithm or workflow complexities

There are two things to consider when thinking about the cost implications for CI/CD as part of the DevOPs projects that require an estimate. In many cases, a CI/CD Pipeline is available at an organizational or department level and its initial deployment costs and maintenance costs are not always considered project costs. The majority of this cost discussion is based on the assumption that the initial investment costs are not part of the project costs (though potential drivers for creation and initial deployment of the pipeline will be discussed briefly for completeness).

The first thing the estimating team needs to discover is how well entrenched the organization and the project team are with the CI/CD Pipeline and the DevOps Culture. Teams that are new to CI/CD or are working on a brand-new project are unlikely to achieve the productivity and quality gains of a team with experience with the Pipeline and that are already engaged in the project. Appropriate questions to ask?

- How many CI/CD projects has the DevOps team worked on?
    - If the team is brand new to CI/CD then:
        - Investments in training will be part of the project costs.
        - The learning curve for adoption will be steep, making initial productivity gains limited.
- How long has the DevOps team been working on this project being estimated?
    - If the project is brand new there other cost considerations:
        - Investment in development and verification of automated test suites
        - Investment to prepare the test data.
        - Investment to prepare the test environments.
        - Cost associated with any additional infrastructure, tools, or other technology that will be specific to the project (additional servers, additional licenses, additional cloud services, etc.)
- Is this project taking an existing legacy project and moving it into the CI/CD pipeline?
    - If the answer is yes the cost considerations are:
        - Investment to refactor features being converted.
        - Investment to create or refactor automated tests (if there are automated tests)

Regardless of whether a project is waterfall, agile, DevOps/DevSecOps or some hybrid, many of the cost drivers don't really change. There are however some agile/DevOps specific project characteristics that

should influence how cost driver values are interpreted in the context of a software estimation model (TruePlanning, Seer-Sem, COCOMO, home grown spreadsheet, etc.) used to perform an estimate. The following cost drivers should be considered or added to the cost considerations:

- Level of automation – Even with CI/CD, there are some tests that will need to be addressed manually. This value should influence the amount of effort for Unit Test, Integration Test and Validation Test – higher automation reduces testing effort. Questions to ask:
  - Automated test coverage (%)
  - Quality of automated testing tools (amount of manual intervention to run and interpret the automated test sets)
- Project Complexity – projects that have more complex logic or workflows will have increased effort to develop and maintain the automated tests and will require more effort to manage the pipeline, they should be rated as more complex than a project the same size (number of features) with less complexity.
- Programming Language(s) being used on the project – some programming languages make it easier to interface with automation tools than others.
- On-going costs associated with infrastructure resources , tools and technologies required for the project (licensing, support, maintenance fees, etc.)
- DevOps Team Size
- Integrations necessary with Third party software applications. Question to ask:
  - How many third-party or legacy applications need to be integrated with the CI/CD Pipeline?
  - What is the complexity of these integrations?
- Training or Retraining requirements. – It is important that a DevOps team regularly get training to keep up with tool and technology changes and to be part of continuous process improvement
- Security Requirements. Questions to ask:
  - Security Level (EAL, NIAP, etc.)
  - Security Process Level – formality and entrenchment of security process

The research of Dora and *The Accelerate Book* did not specifically cover the cost implications of DevOps, CI/CD and Automated Testing but it certainly provides the software estimating community with some things to consider when estimating software project where DevOps and CI/CD are in place. By approaching the DevOps organization to determine:

- Deployment Frequency
- Lead Time for Change
- Mean Time to Recovery
- Change Failure Rates

Using this information and the numbers from the 2022 State of DevOps Report (Figure 2 repeats this information in a tabular fashion from the report [2]) the software estimator should use the information from the team to make determinations for the best values for the following additional cost drivers:

Software delivery performance metric	Low	Medium	High
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	Between once per month and once every 6 months	Between once per week and once per month	On-demand (multiple deploys per day)
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Between one month and six months	Between one week and one month	Between one day and one week
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Between one week and one month	Between one day and one week	Less than one day
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	46%-60%	16%-30%	0%-15%

Figure 2 - Software delivery performance metrics (Source [2])

- Organizational Productivity – this value represents a comparison of the overall productivity of an organization's ability to deliver software efficiently to other organizations that develop the same types of software: Questions to ask:
    - How well has the Organization embraced a DevOps culture. In other words, how supportive and cooperative is the organization with respect to DevOps across it projects (low, medium, high). Low DevOps Culture would make an organization less productive than a similar organization that is in embracing the culture.
    - How low is turnover for the DevOps Teams
    - Team Score on the four DORA key software delivery performance metrics.
- DevOps Team skill and experience: Questions to ask:
    - Experience with the Project and Programming language(s) (years)
    - Team Continuity
    - Familiarity with the CI/CD Pipeline
    - Team Score on the four DORA software key performance metrics.

While one should not ignore the fact that organization's that invest in creating a DevOps culture with good CI/CD Pipelines will need to spend money on tools, training and the learning curve associated with these, there are cost benefits that should be realized with this investment when done properly and thoughtfully.

- Fast and frequent released software can impact cost by reducing deeply expensive downtimes due to major defects being detected post-production. This impacts not only the bottom line, but it can impact the organization's credibility as well.
- The quick turn-around associated with frequent integrations and deliveries means that the process of troubleshooting problems is significantly streamlined – freeing developers up to spend more time on creating new features.
- New team members can more quickly come up to speed with DevOps teams operations since many tasks have been automated.

- Freeing developers from manual, repetitive tasks and allowing them to put more creative energy into their jobs tends to keep them happy and more likely to stay with an organization. It also increases the ability to deliver new features to the users, keeping them happy and engaged and renewing their licenses. This potential benefit really highlights the important difference between having tools and automation and creating a DevOps culture.

In the event that the project estimate is required to include the costs of the creation and implementation of the CI/CD Pipeline. For completeness, the cost considerations will be briefly discussed here:

- Infrastructure may need to be purchased to support the new tools and technologies required for the pipeline including costs of servers, cloud services, network equipment, etc.
- Tools may need to be purchased, if they are not available, for version control systems, test automation, deployment automation etc.
- Training will be required to teach the DevOps team how to build and use the pipeline.
- The time it takes the DevOps team and IT personnel to build and maintain the pipeline – creating tests for the pipeline and addressing operational issues during startup
- The Pipeline needs to be integrated with the other tools and systems within the organization. Depending on the extent and complexity of these integrations, this could represent a significant cost
- On-going costs associated with maintaining, upgrading and scaling the pipeline for the lifetime of the project need to be considered as well.

6. Conclusions

The DevOps approach merges the skills of Development (Dev) with those of Operations (Ops) to optimize software delivery efficiency and customer satisfaction. One important premise of the DevOps approach is to create a culture where an organization embraces the value added to the software delivery process through automation, collaboration, continuous process improvement, rapid delivery and better resource and cost management.

Key practices that enable DevOps are:

- Continuous Integration which creates an environment where integration and automated testing are kicked off as new code enters the system.
- Continuous Delivery which creates an environment where a build is automatically created, verified and deployed into a production or production-like environment
- Continuous Deployment which furthers Continuous Delivery by deploying a build directly to a Production environment after the test suites pass
- Configuration Monitoring involves continuous collection, analysis and reporting of key performance metrics to identify issues and optimize the performance of the software development lifecycle.

More organizations are using the CI/CD Pipeline which is an organized and automated suite of tests that is used to ensure all software versions are up-to date, compatible and ready to use. CI/CD Pipeline allows the process from code check in – to build – to testing – to delivery to be completed automatically. The goal is to automate as many repetitive tasks as possible which frees up developers to focus on creativity and building new features.

CI/CD Pipelines have many potential benefits but these do not come for free. There are costs associated with creating, building and maintaining the pipeline. There are also cost estimating implications associated with the fact that a software project is embracing CI/CD and DevOPs as their software development and delivery process. This paper presents cost implications within the context of an existing software estimating model when CI/CD Pipeline was part of a project. These were presented in the form of suggestions for new cost drivers as well as suggestions on how existing cost drivers need to be re-evaluated to better represent the associated costs and costs benefits of CI/CD practices. Four key software delivery performance metrics were presented to help organizations understand how their DevOps performance compares to the rest of the software industry where DevOps are embraced.

References

[1] "The 2021 State of DevOPs Report" available at https://www.puppet.com/success/resources/state-of-devops-report, Retrieved January 2023
[2] "Accelerate 2022 State of DevOps Report" available at https://services.google.com/fh/files/misc/final_2022_state_of_devops_report.pdf Retrieved February 2023
[3] Armlin, Davc, "10 DevOPs Tools for Continuous Monitoring", ChaosSearch Blog, July 2021, available at https://www.chaossearch.io/blog/continuous-monitoring-tools, retrieved November 2022
[4] Hilton, Michael, et. al, "Usage, Costs, and Benefits of Continuous Integration In Open Source Projects", ASE 2016, Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, August 2016, pp 427-437, available at OpenSourceCIUsage.pdf (oregonstate.edu), retrieved January 2023
[5] Hodgson, Pete, *Continuous Delivery in the Wild,* O'Reilly Media, 2020, available at O'Reilly: Continuous Delivery in the Wild, Retrieved February 2023
[6] Lau, Grace, **"**Modern Mainframe Development: A Comprehensive Guide to the Best CI/CD Practices", IEEE Computer Society, May 2022, available at https://www.computer.org/publications/tech-news/trends/the-best-ci-cd-practices - Retrieved February 2023
[7] Bigelow, Stephen, "CI/CD Pipelines explained: Everything you need to know", Tech Accelerator, May 2021, available at CI/CD pipelines explained: Everything you need to know | TechTarget – Retrieved Feb 2023
[8] Fedak, Vladimir, "How CI/CD Automation Saves Time and Money", itsvit, March 2021, available at How CI/CD Automation Saves Money and Time | IT Svit, Retrieved January 2023
[9] Boyko, Oleg, "Getting Started with Continuous Monitoring", DevOps.com, March 2021, available at Getting Started With Continuous Monitoring - DevOps.com , Retrieved November 2022
[10] Ali, Junade, "The Accelerate Book, The Four Key DevOps Metrics & Why They Matter", Haystack, January 2023, available at The Accelerate Book, The Four Key DevOps Metrics & Why They Matter (usehaystack.io), Retrieved February 2023