Tampere University

Ahmed Gebril

CONTINUOUS INTEGRATION, A LI-TRERATURE REVIEW

Engineering and Science
Bachelor's Thesis
April 2020

ABSTRACT

Ahmed Gebril: Continuous integration, a literature review.
Bachelor of Science and engineering thesis
Tampere University
Science and Engineering
April 2020

Adopting Continuous integration (CI) and continuous delivery (CD) has become a powerful approach to help software engineers integrate, build, and test their work more frequently, resulting in faster deployments in the production side. The CI/ CD approach works within automating tools that checks the code correctness according to a setup in a software engineering pipeline before integration. Adopting CI/ CD could have numerous advantages; however, there are challenges.

This study aims to identify the benefits and challenges of adopting CI/ CD, associate these benefits and challenges to the stages within a continuous delivery pipeline (CDP), and explore whether these benefits could encourage decision makers in an organization to adopt CI/ CD.
Most common challenges, which could discourage product owners from adopting CI/ CD approach, along with recommended resolutions collected from literature are listed and demonstrated.

The findings indicate that adopting continuous integration has numerous benefits that the challenges associated could be resolved with some effort and resources allocated. A total of four main benefits, and four main challenges with proposed solutions to them where found. It is also indicated that organizations are to make analysis whether these challenges could be managed and consequently decide on whether to adopt CI or not. There is a lack of research on expected cost of adopting CI/ CD and further on-depth research regarding actual costs would be needed in the future.

Keywords: continuous integration, continuous delivery, continuous deployment

PREFACE

I would like to thank my employer Eija Hartikainen for suggesting this thesis topic as it opened me a door to understand a new aspect of my workplace more deeply.

I would like to thank my supervisor, Terhi Kilamo for her support and guidance throughout the writing of the thesis. I would also like to thank my family and friends for providing the support and motivation for completing this thesis.

Tampere, 20th of April 2020

Ahmed Gebril

CONTENTS

LIST OF FIGURES

.

1. INTRODUCTION

Software development incorporates various methodologies in developing and testing circles. The most used of these are continuous integration and continuous delivery, Agile, an approach that defines iterations to software development and project management [41], and Development and operations (DevOps). Though DevOps, agile, CI/ CD are different, they complement each other [1].

Software development methodologies serve as skeletons for defining tasks performed at each step such as requirement gathering, analysing, designing, implementing, testing, and maintaining in the software development process [2]. Agile acts as a chain connecting the gaps in communication between developers and customers [1]. Agile focuses on the development process as it provides an alternative to traditional sequential software development through employing customer collaboration and a whole team approach [3]. CI/ CD holds the uttermost importance when adopting agile methodology; in other words, it is an agile best practice [4].

DevOps is a culture; it helps in providing a blueprint for the stages used in CI/ CD [5]. Serving as a chain connecting IT operations and developers, DevOps helps in breaking down the barriers between engineering and operations; it offers cross training about both side's skills. That leads to an overall increased ability in participating in each other's tasks and more high-quality collaboration and robust communication [1]. CI/ CD focuses on technical practices highlighting tools used in the automation process.

While there have been instructions on how to adopt CI/ CD, there has not been a lot of research done on what challenges could be associated with adopting such a discipline; this study aims to investigate what these challenges could be and whether the benefits outweigh such challenges [10]. The research question in this study is what the pros and cons of adopting CI/ CD are, throughout studying the stages of the methodology, hence, associating the benefits and challenges with each possibly used stage. This study could be useful for developers for understanding how useful implementing CI/ CD could be and what kind of challenges to expect with the implementation, besides knowing proposed solutions for such

challenges. The literature is collected from official documentations on the technologies used in the stages and from previous studies, articles, and research papers.

The rest of the thesis is structured as follows: Chapter 2 explains the difference between CI and CD, and examines the most common stages adopted by organizations already utilizing CI/CD, from the build stage to the deployment stage to the production side, Chapter 3 lists most discussed benefits and challenges associated with adopting CI/ CD, Chapter 4 serves as a general discussion on the findings and aims at finding solutions for the most reported challenges; Chapter 5 concludes the study.

2. CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY

Continuous integration and continuous delivery (CI/ CD) are a coding methodology and set of practices. Once developers within the same or different teams are done with implementing small changes, they need to commit these changes to version control repositories frequently. Developers in different teams, sometimes even in the same team, are mostly using different platforms, tools, and technologies. The team needs a mechanism to integrate and validate these changes. Testing in Continuous integration expands beyond utilizing automated testing frameworks, which can help development teams and quality assurance engineers know whether a software build passes or fails [6].

CI/ CD is a growing field of interest in the industry. As for being competitive in today's market, corporations are to launch new features that matter to their users faster than their competitors and getting quicker feedback on the work is crucial to achieve faster deliveries. According to Fowler "The whole point of continuous integration is to find problems as soon as you can and receive frequent feedbacks" [25]. It is up to decision makers in corporations to decide on whether to adopt CI/ CD or not. However, this is a question that could be answered by investigating the pros and cons associated with utilizing such a methodology.

As mentioned earlier, Agile is just an iterative approach focuses on the development process supporting CI/ CD in achieving one of its main goals which is receiving quicker feedbacks. When an organization is lacking CI/CD as **Figure 1** shows, there is going to be an extensive process required at every stage of the software delivery cycle. The organization is presumably not Agile [30]. And with such, delivering the product to the customer would take longer than expected, as there are always extra and unexpected layers of approvals added. Even the smallest software changes would not be an exception.

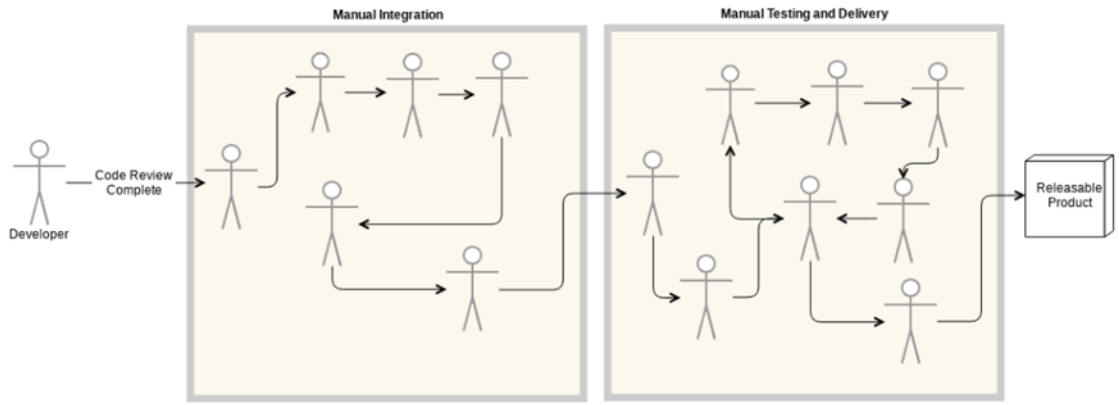

Figure 1. Integrating, testing, and delivering a software without CI/ CD. [30]

What CI/ CD aims at, as shown in **Figure 2**, is continuously receiving feedbacks for small sections within the project and eventually on the whole project. One of the most common frameworks for Agile is Scrum, where the project is broken down into features and these features into user stories, which are continuously integrated [31].

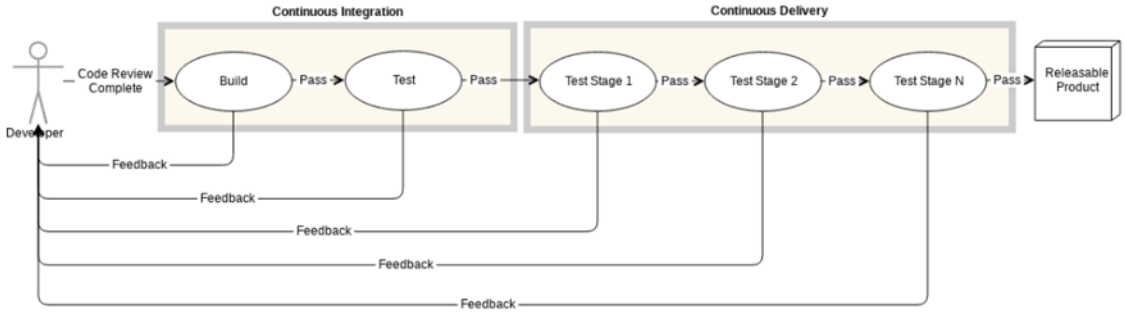

Figure 2. Integrating, testing, delivering a software product with CI/ CD *[30].*

Even though CI and CD are often mentioned together, they are not synchronous. CD might either refer to continuous deployment or continuous delivery.

2.1 Continuous Integration

CI puts emphasis on building, integrating, automatically testing, and packaging the ap-plication, helping with checking that the application will not break whenever

any changes are made to the code, and allowing a release of the application whenever requested. With consistency in the integration process in place, teams are more likely to commit code changes more frequently, and that leads to a better quality in the code and quality assurance. [6]

Testing the code frequently is a prerequisite in CI; the goal is to deliver quality products to the customers. Continuously testing a code is often implemented within a set of auto-mated regression, performance tests in the CI pipeline.

Teams adopting CI may opt to deploy their codes to the production side as frequently as they wish, on a daily or even hourly basis.

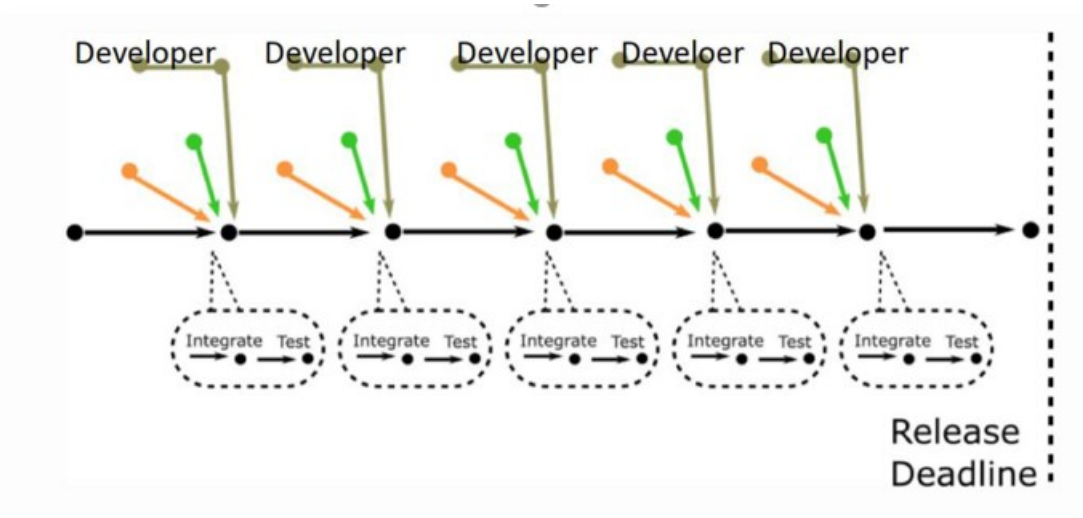

Figure 3. Embodies frequency in integrating codes in CI. Features are integrated with small increments instead of waiting before an official release is announced. [7]

2.2 Continuous Delivery

CD starts from the endpoint of CI. CD allows manual release of an application to certain infrastructure environments. Teams within the same organization are often working with multiple environments other that of the production, and CD helps in ensuring that the developed application will not break whenever any code changes are pushed. The code is manually delivered to the production side whenever needed [8]. The goal of continuous delivery is not necessarily to deploy the result, but to ensure that the result is deployable. The mechanism of continuous delivery is the continuous delivery pipeline (CDP). Nevertheless, CD is not always the optimal solution for delivering an application to the customer [9].

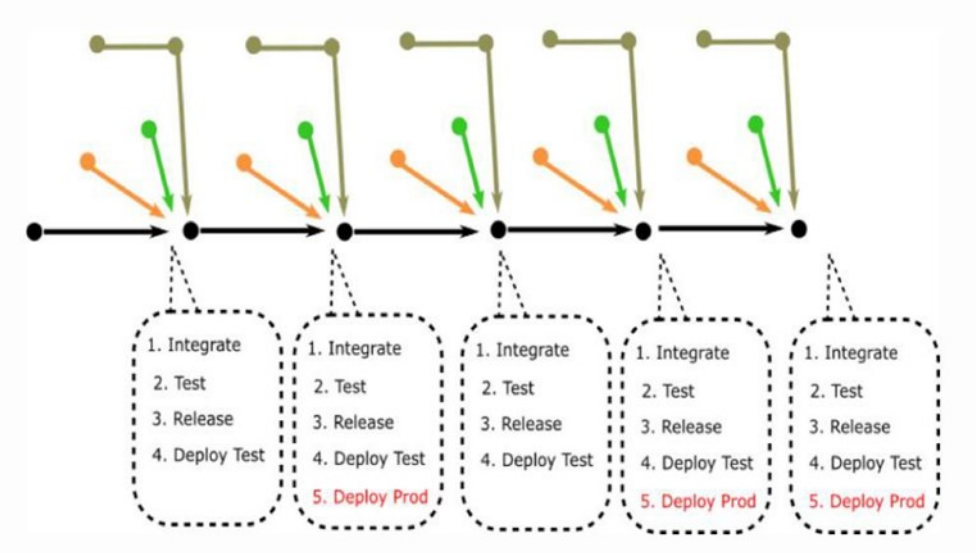

Figure 4 shows the steps followed by the pipeline in CD, which takes CI further by a button click. [6]

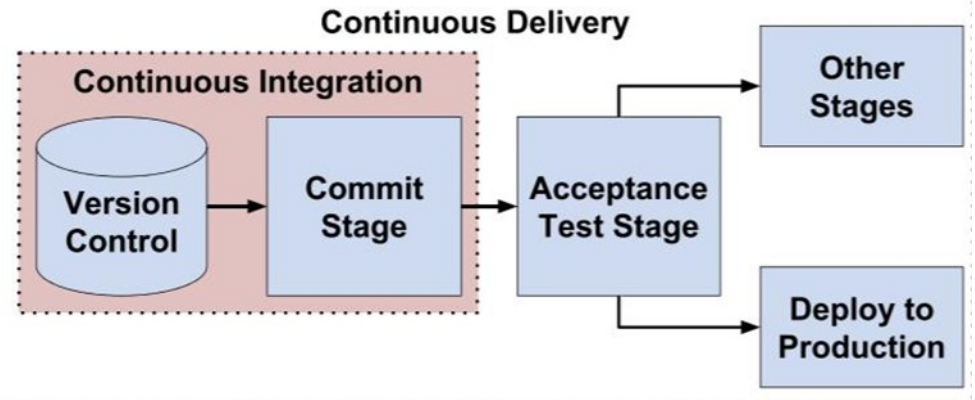

Figure 5. The conceptional difference between CI/ CD. [10]

2.3 Continuous Deployment

At this level everything is automated, human intervention is not needed. All the changes passing the stages in the pipeline are released to the customer. CD provides a regular and continuous feedback from customers and allows all the work done by developers to be continuously reviewed. Not all corporations integrate CD as it is considered an advanced phase of delivering the product. [31]

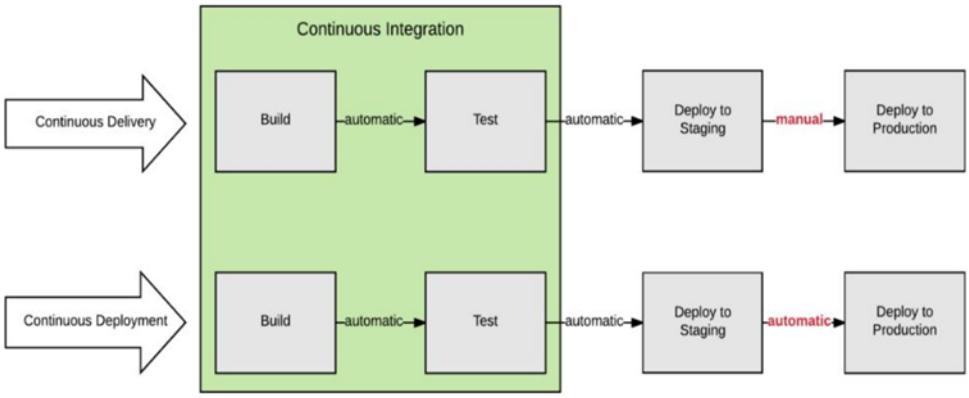

Figure 6. The difference between continuous delivery, where deploying is manual and continuous deployment, where deploying to production is automatic. [12]

2.4 Stages of the CI/ CD Pipeline

Figure 7 depicts the stages within a continuous integration pipeline, viewed from the perspective of adopting them on integration servers such as Jenkins, Travis CI, Bamboo. The actions taken during each step are written in a build script. The stages achieving the desired outcome of mainly getting quick feedback on the code and required to maintain the health of the application, are described conceptionally. The stages in the pipeline are stated in a configuration file, which allows a customization of the stages that the application could go through.

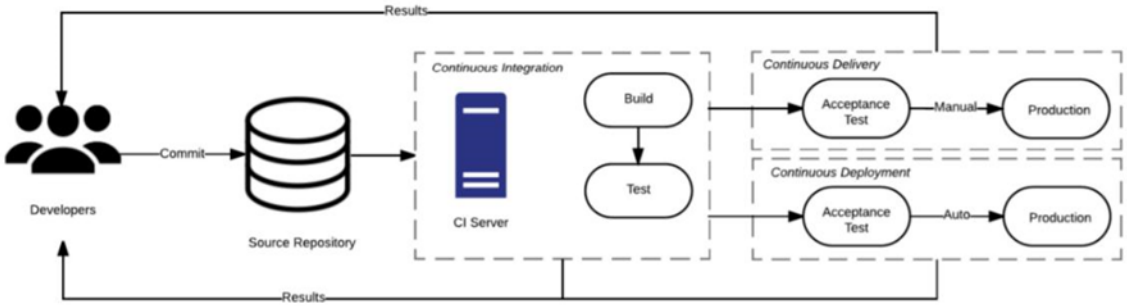

Figure 7. Stages of continuously integrating and deploying a software engineering product [10]

Frequent feedbacks are achieved from frequent builds in a CDP. Every time there is a code change, the Repository, where the codes are pushed to, notifies the CI/ CD server. This triggers CDP for building and testing the application, and if all configured stages are passed, it is ready to be deployed in the Artifactory. As depicted in **Figure 8**

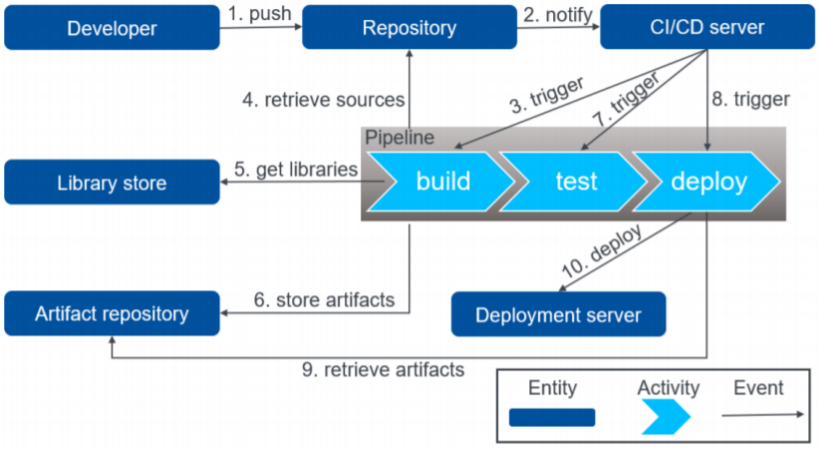

Figure 8 *Workflow of integrating code into a CDP, building the application is a repetitive process.[44]*

2.4.1 Version Control

Developers continuously commit their changes into Version Control Systems (VCS). VCS help us track all changes committed so that they do not get lost. Nowadays, Subversion and Git are the most widely version control systems. [13]

Subversion is a centralized version control system (CVCS), Git is a Distributed version control system (DVCS). CVCS has a separate server and client. Developers using CVCS have control only on the files they are working on, whereas DVCS focuses on sharing code changes, allowing developers to have full control on the project on their local machines by having a full version history. [32]

CI requires that the codebase must be under version control. Every change applied to the codebase must be safely stored in a dedicated VSC. Once the code is in version control, it can be accessed by the CI tool.

Git is a powerful VCS, and its characteristics serve CI in many ways, some of these are:

• Huge branching capabilities: Feature branches provide an isolated environment for every change in the codebase. As discussed earlier the first step in the deployment process is committing the code changes. That cannot be done without creating a new branch, no matter how big or small the change is. This ensures that the master branch only contains code that can be used in the production environment [14].

• Distributed development: Every developer can access the whole repository including its history and can work on it locally.

• User Authentication: Changes are traced with their commit ID and their author identity.

• Efficiency in handling large projects: Performance testing, which is a testing practice that aims to determine how a system behaves in terms of stability under a particular workload, indicates that Git is fast.

• Garbage Collection: Git optimizes the memory usage by freeing up disk space and packing similar objects together. [33]

Once changes are committed to a version control system, the pipeline assembles the application, in other words compiling the application and pulling the dependencies, which are anything needed for the application to work, such as modules or libraries. Build stages allow us to run applications in a sequential order and multiple parts of the application in parallel. [15]

2.4.2 Code Review

When a single developer is working on a project, they could commit their changes by pushing them to the master branch in the VCS [16], or a dedicated branch. However, a team working on a single project need to use a different approach, which is code review; the code review process should not consist of one-sided feedback. Code review is essential because it does not only check if the code is free of bugs but also, confirms that the code adheres to certain guidelines, i.e., styling, functionality. There are multiple ways of reviewing the code, email pass-around, pair programming and Gerrit code review, the latter is one of the most popular nowadays. [17]

Reviewing other developer's work is done in Gerrit, where the last version of the code is stacked alongside the changes made by the developer. This approach is called Side-by-side patch view as depicted in **Figure 8**.

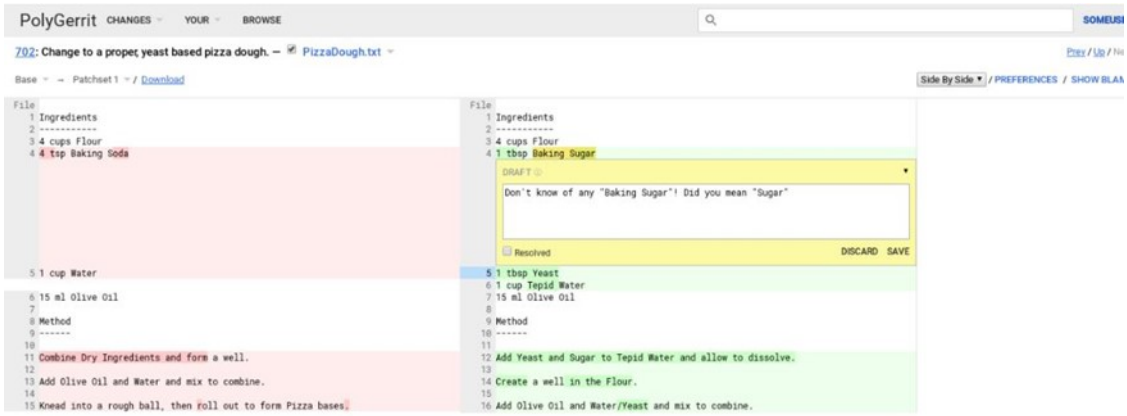

Figure 9. Side-by-side patch view used in peer reviewing in Gerrit. [17]

A review is done through voting, where the highest vote is +2 and the lowest is -2. Once the change achieves a +2 review by two different developers, mostly seniors, it moves into the CDP for testing. The pipeline tool detects the changes made in the source code repository, pulls these changes, and starts a new build. If the build succeeds, the changes get merged into the master. Otherwise, the team responsible for the changes will be notified. The pipeline tool will continuously keep looking for changes made in the source code [18].

2.4.3 Unit Testing

Adopting CI requires intensive testing; ensuring that the product can be reliably shipped to the production side requires various stages of testing, starting from small changes to the final current version of the whole product.

Unit testing, where discrete components of a software are tested, is the first step in testing a continuously integrated software. Usually based on a single input, unit tests are meant to validate that each unit of a software behaves as expected. I.e. testing the expected behaviour of a function against what it results. It is essential that unit tests are written throughout the development process as it would be very difficult to trace a bug if a discrete component of a software is not tested individually. Unit tests are usually conducted by the same developer who wrote the code, making them simpler than other types of tests. [45]

Unit tests should be quick as they require no external resources, and because of their pace and coverage of small units, besides requiring low maintenance, and therefore being cost-effective, they could be viewed as the first line of defence against bugs. Because a small bug would inevitably result in defected application when deployed to the production side, unit tests are very important to the overall health and maintainability of an application. [19]

2.4.4 Code Quality Analysis

Analysing the code in terms of its security vulnerabilities and against a set of coding rules ensures the codes readability and maintainability. Code quality is crucial as it impacts the overall software quality. [20]

A high-quality code can do what it should, follows the expected behaviour, follows a consistent style, is easy to understand, includes proper documentation, and is testable. Codes are best analysed with automation tools that run static analyser over code early and often.

One of the most popular platforms to run automatic static analysis is SonarQube, an open-source platform [21].

SonarQube examines the code based on several criteria such as: complexity of the code, number of lines, vulnerabilities, false positive issues and more [37]. If the code fails to comply with its rules, it raises an issue.

SonarQube has an exhaustive list of metrics, some of them are: [34]

- Reliability: Examining bugs in the code and an estimate effort associated with fixing such bugs.
- Security: Examining vulnerabilities, their severity, and the effort associated with fixing them.
- Maintainability: examining code smells and technical debt (TD), which is the effort associated with fixing all maintainability issues.

SonarQube analyses the code according to four main types of issues: [36]

- Code Smell: A maintainability issue that could possibly suggest a deeper problem in the application.
- Bug: A maintainability issue that could result in breaking the code.
- Vulnerability: Security issue that puts the application at risk and as such needs to be fixed immediately.
- Security Hotspot: Highlighting a security-sensitive piece of code that needs to be reviewed, upon which, it is determined whether there is either no threat or that fixes to the code must be applied.

2.4.5 Integration Testing

Integration testing is a level at which individual units of a software application are grouped and tested as one entity. This level of testing helps expose bugs and faults occurring with the interaction between unit tests to verify functionality of unit tests together [22]. Integration tests can also quickly detect issues happening when an application communicates with an external system. This includes external resources such as networks, databases or any third party components .This

is an important step as the customer might be using extra components and that would require the developers to test how the application interacts with such components. Integration testing broadens the test coverage and improves the reliability of tests.

There are two main types of integration testing, one of them is Big Bang approach, at which all unit tests are merged and tested at once. Another is incremental integration testing, which is divided into three types, Top-Down, Bottom-Up, Sandwich integration tests. Incremental integration tests are performed by connecting two logically related modules and testing them together and incrementally increasing the test coverage till all modules are tested. [46]

Implementing integration tests are time-consuming as it requires a lot of setup. The tests are difficult sometimes because they depend on external factors such as platforms, environments, and databases. And there might be less compatibility between two systems developed by two different corporations. Choosing the appropriate approach for integration testing is quite challenging because there are advantages and disadvantages of choosing each. However, implementing a solid set of integration tests is one of the best things you can do to ensure the long-term stability of your application.

2.4.6 System Testing

System tests are carried out to ensure that the whole software system is compliant with the requirements and specifications. Therefore, they require a fully installed system. Such tests ensure that the external interfaces such as Graphical User interfaces (GUIs), and Web page endpoints, work end to end as expected. Systems tests are usually lengthy in runtime besides having extensive set-up times.

The key to having successful system testing is having proper initial unit testing. This eliminates the risk associated with faulty components within the whole software, resulting in faster and more efficient system tests [47]. System testing is performed by a testing team rather than the development team. That ensures having an independent work platform.

There are four types of system testing: [50]

- Performance Testing: Determines how the system performs in terms of speed, stability, and reliability.
- Load Testing: Determines how the software responds under extreme loads.
- Stress Testing: Determines how the software responds under different loads in terms of intensity.
- Scalability Testing: Checks how the software performs in terms of scaling the user request load up and down.

2.4.7 Acceptance Testing

After individual codes pass previous stages of testing, they are integrated into the pipeline for acceptance tests, which is the stage of testing the whole system for acceptability. The main purpose of acceptance tests is ensuring that the final product complies with the requirements created by the customer. And verifies that the product meets the quality standards of agreed upon between the customer and the product owners [23].

Acceptance tests holds the role of evaluating the system in a production-like test environment. This gives an insight on how an external user perceives the system [38].

One prominent aspect of making acceptance testing efficient is test driven development (TDD), which is a development technique where writing tests comes before the code implementation by writing a test that fails and the only way to make it pass is by writing a proper code implementation [38].

Acceptance criteria come in different suites: [47]

- Functional: Testing the functionality of the application.
- Non-functional: Testing the application for criteria such as security, maintainability, reliability, performance.

Acceptance testing come in types some which are:

- User Acceptance Testing (UAT): Also referred to as, End-User Testing, where the product's functionality is tested for verification from the end user' perspective.
- Business Acceptance Testing (BAT): Determines whether the product complies with the business goals in terms of both functionality and non-functionality. As they focus on business profits, BAT could be quite challenging because they should always meet the changing market conditions.

2.4.8 Artifactory

Codes that survive all the previous stages can be merged into the master. However, keeping versions of such releases provides a library at which any version of the application can be dispatched whenever considered ready [24]. This allows making modifications easy whenever the customer requires changes in the application. The VCS tags the version either as a patch release, which could refer to a regular release for the product i.e., minor new features, bug fixes or enhancements. Or as hot-fix, which is a reported issue by the customer that is needed to be fixed as soon as possible, a hot-fix is released on request, whereas a patch version is released at regular intervals [35].

Artifactory is similar to source code repository. An artifact repository includes artifacts that are the result building the application. It stores metadata, which is data needed by the application and described by another data, such as versioning and dependencies. [48]

Keeping the application's versions in the repository has numerous advantages in a CDP such as: [48]

- Highly available and stable systems: As DevOps aim to, the system should be kept running. As it is crucial that the system could be deployed in high availability whenever needed.
- Managing many binaries of the system across different platforms: Only one copy of the system configuration, which is referred to as binary, is stored, and the copy could be accessed by different teams. Therefore, there is no need for replicating extra environments to support the development cycle.
- Security, Access Control and Traceability: The binary itself could be accessed with different access rights among teams, mitigating any security vulnerabilities. I.e. restricting access to reliable third-party resources that are already approved.

3. BENEFITS WHEN ADOPTING CI/ CD

There are numerous benefits associated with adopting CI, from the build to the deployment stage. A developer can debug their codes more easily, as errors can be associated to small increments in code, fixing these small incremental errors prevents accumulating these errors into a big chunk of bugged code [25]. Merging the code frequently helps in avoiding merge conflicts [26]. CI allows finding likely problems that might arise during the development stage earlier and that improves product predictability [27]. The overall result of all these benefits is increasing the productivity by reducing the time spent chasing and correcting these integration bugs.

3.1 Frequent Builds

The fundamental benefit of CI comes from removing lengthy sessions at which people spend time hunting bugs without even knowing who was responsible for this bug happening [32]. Frequent builds allow developers to catch bugs earlier by making small segments of the code. That makes developers less worried about breaking their builds and let them spend less time debugging [28].

As the first phase in CDP, the build phase constantly and incrementally merges code changes along with testing and security validation. Frequently building an application is not inclusive to compiling it. Building an application is followed by all the other phases stated in the pipeline configuration such as testing and code quality analysis.

3.2 Testing in the Cloud

Modifications in the code could be constantly saved in the cloud, an online hosted environment that allow controlled setup of environment, freeing up space in personal machines [29]. Developers and testers can trace back a change made to the code with its date and author of the change. Tracing these changes make corrections much easier and faster. Hence, developers would not keep tons of copies as a backup whenever there are modifications to the code just in case a build or a test fails, or something goes wrong.

Cloud testing allows controlled conditions that could clear out any imminent problems in the production side when deploying the application. For that the environment in both the development and the production sides are to be identical. [51]

Cloud testing benefits are not only exclusive to memory optimization but could also include providing scalable services and improved collaboration. [52]

3.2.1 Scalable Reinforcement

The cloud allows the application to be scaled in volume, capacity. The cloud provides platform-as-a-service (PaaS), which is a whole set-up development environment in the cloud with the customized resources that could serve from simple applications to sophisticated ones. PaaS includes an infrastructure for applications to run on, servers, customized storage, databases. Therefore, PaaS reduces the expenses and complexity of buying software licences separately. [53]

3.2.2 Improved Collaboration

Because of its nature in allowing resources to be globally available. Cloud Testing unifies development procedures, complying with DevOps principles that aim towards continuous collaboration among project participants, facilitating the process of continuously delivering the product releases. The Genuity of testing through the cloud assures overall high quality for the application.

3.3 Faster Deployment

A mature pipeline would allow automatic deployment of the code to the production if all build tests pass. Unlike the pattern followed by CI would depend on small changes, developing the code as big chunk means that the code would need to be tested once all developers finish their small increments. And that leads to inevitable bugs and therefore delays in deploying the product to the production side.

CI helps us ensure that code works in the production side since the development environment is ideally matching the production environment [7]. Aiding in avoiding the hassle of testing and chasing bugs after the program is thought ready to be deployed steering clear of the long sessions of testing that would cause customer frustration.

Keeping binary versions in the Artifactory keeps the product in a deployable state, with just a simple button click, a requested version of the application is ready to be shipped.

3.4 Quality Assurance

Frequent deployments would mean having frequent communication between the developers and the customers, even if not directly, and that allows more frequent and faster feedbacks [30], which is the whole idea behind Agile and DevOps. Hence lowering the risks of developing unnecessary parts in the project. And that ensures that things would not go wrong due to miscommunication. Continuous delivery aims to keep a product continuously in a releasable state. Thus, if the end user requests a release at any given time, it should be reliable. Raising the overall quality of the process [30].

Quality assurance is categorised in three main roles [39],

- Testing: Packing a set of tests such as Integration tests, helps developers access these tests whenever needed, assuring the code quality in the long run.
- Delivery: Deliveries can be done automatically or deployed whenever needed.
- Optimization: One of its aspects is making the code self-documenting. This is huge aspect in assuring the quality of the work. Unit tests help developers understand the expected behaviour of the code without actual written documentation.

Quality assurance also refers to the overall quality of the code tested by Static Analysis tools, which as indicated earlier, help in testing the quality of the code in terms of security, maintainability, and reliability.

4. CHALLENGES WHEN ADOPTING CI/ CD

4.1 High Initial Cost and Effort

Setting up the CI pipeline can be time consuming [30]. Initial adoption of CI requires a huge effort for setting up the right environment that would ensure proper building and testing. The development environment should match the production environment and that also requires hardware resources needed for the test environment to mentor the performances of the product in different situations. The perceived initial effort for adopting such CI system could discourage product owners from the adoption [10].

4.2 Testing Automation Related Challenges

One prime challenge in test automation is that it requires a lot more initial effort than manual testing. Lack of proper testing might lead to broken code in the production side. Therefore, making sure that the tests are thorough is not a simple task. Automated tests also require skilled testers or that the existing testers would be trained, besides having the skills needed for adopting new tools [30].

Not all products or their components can be tested, a complex software that has multiple dependencies that are implemented in a way that it would accept testing requires a lot of communication to have a thorough understanding of what makes the application testable. Another type of mostly reported test fails are flaky tests, which are tests that fail sometimes even though there might not be bugs in the code. Such tests are time-consuming and could result in a lot of frustration to the developers [10]. Another common problem is UI testing bearing in mind the fact that the user interface of the application is the part that undergoes changes most frequently could cause unexpected outcomes in the test environment.

4.3 Integration Related Problems

Integrating the code changes regularly are undoubtably very helpful for tracing the changes. However, it should be done in a proper way. A lot of problems come from broken builds, large commits, and merge conflict, which arises when different developers make changes in the same line, for instance, one developer edits a file, and another deletes the same file. [10] Reports 7 reasons behind integration problems. The two most critical ones are:

- Broken builds: become problematic when a build breaks, fixing and maintaining such build takes a significant effort.
- Slow integrations approval: become problematic when integrating changes becomes a lengthy process by strict approval processes, for instance approval by a project manager.

4.4 Security Related issues

One major challenge in implementing CDP is dealing with security issues. The availability of the application components and its dependencies on external tools could create risks. Some of these risks are: [54]

- Security Risks in servers: Servers accessing the application could jeopardize the applications security. Such issue is prominent when testing on the cloud.
- Security Risks in CI server: Since CI servers could be accessed from anywhere. There is a risk of modifying or deleting the CDP if there are no proper access right mechanisms implemented.
- Security Risks in the VCS repository: User authentication using password is the only way of securing access.

5. DISCUSSION

5.1 Frequent Builds vs. Integration Related Problems

Since we used as the example VSC in this study; it is obvious that Git serves CI/ CD pipelines in tremendous ways, such as creating a branch that holds a specific change before the change gets accepted and merged. Yet since some issues might arise when developers try to push these changes, for instance, merge conflicts, integrating small changes and constant communication between developers is crucial. And in case integrating some changes would result in broken builds, the fastest fix is reverting to the mainline [25]. [10] Discusses in details the reasons behind broken builds and proposes three solutions to them.

- Rejecting bad commits: a practice by which commits that automatically fail prerequisites and standards, i.e., failing some tests are rejected from merging, keeping the branch always functional.
- No Branches: Keeping only one main branch to contain all the code changes and no other branches are allowed prevents possible problems caused by long-running branches.
- Monitoring build length: keeping the build length as short as possible and taking actions if the build gets lengthy.

5.2 Implementation vs. Initial cost and effort

As discussed earlier, implementing a CI system would have numerous advantages in general. According to [42], In 2018, 38 percent of organizations that undertook DevOps practices along with CI exhibited a revenue growth of 10 percent more than that of prior year. On the other hand, only 25 percent of those who have not adopted CI had a comparable growth.

According to [10] The initial effort in implementing CI refers to:

- Resources: Effort in fixing and maintaining broken builds, Insufficient hardware resources, network latencies that could cause delays in builds.
- Human and organizational factors: lack of motivation, lack of experience, lack of discipline.

And there are specific recommendations on resolving such issues

Human and organizational factors could be mitigated with these practices:

- Situational help: providing help needed by members in related situations aids in mitigating lack of experience of these members.
- Training: Organizing sessions for the team to be comfortable with implementing a CI/ CD system and maintaining it could mitigate the lack of experience.
- Demonstration: Constantly exhibiting to team members how important undertaking DevOps practices along with implementing CI/ CD would benefit both individuals and organizations.

Resources issues could be mitigated with these practices:

- Tooling: providing enough tools that would be able to provide a an easy-to-follow feedback for all the stages in the pipeline.
- Providing hardware Resources: providing enough hardware resources that could simulate a production-like environment.

Since cost might be a hindering factor for organizations to choose all tools and resources needed for implementing a CI system. Making a cost analysis is essential for the organization to decide which tools and resources to choose.

5.3 Multiple stages of testing vs. Testing Automation Challenges

The fact that a snippet of code goes through unit testing, the whole application through integration testing, and the application's quality is assessed from an end-user perspective, helps ensure that the code is free of bugs and complies with the quality assurance.

However, the issues mentioned earlier could be a challenge in the pipeline. In this section some recommendations are proposed for these issues:

5.3.1 Flaky tests

Tests that pass or fail periodically without change in the actual implementations are tricky and cause a lot of issues. [43] lists ten categories of flaky tests and discusses three of them in details:

- Async wait: such problems occur because the test and the application under testing are running in separate processes. For example, a test that is expecting a response from a server is not properly synchronized to wait for the server to return a response. This can be resolved with configuring the test to wait for a specific period before the response has been received.
- Concurrency: such problems occur because of a non-deterministic nature of the test. Such as a code that can produce different valid outcomes, yet the test is expecting an exact outcome. This issue could be resolved by making a more generic test that could accept all valid outcomes.
- Test order dependency: such problem occurs when tests depend on frequently changing data. Such as adding data to a database. This can be resolved by rolling back the database to the previous state after the test finished executing.

5.3.2 Complex software with multiple dependencies

Testing a software gets more challenging when it depends on a lot of external resources. This might cause the tests to be slow, a problem in Agile practices,

especially if there are expected release dates for the software. Some recommended practices for speeding up tests are:

- Containerization: Using cloud technologies to package a software into Standardized tests for development, shipping, and deployment. And bundling the code with its minimum requirements needed to run. Such approach also helps running the tests in parallel, resulting in higher speed and efficiency [40]
- Modular Testing: breaking down the application into small functionalities. Such practice helps creating a road map that is easier-to-follow.

5.4 Security Validation vs. Threats

Enhancing security mechanisms in a CDP is essential and could be achieved by several methods including: assigning different roles and having strong authentication passwords for the VCS. Team members should be granted different levels of access. It is also crucial for servers in which CDP could depend on to be securely configured. And totally isolating testing and production environments [55]

5.4.1 DevSecOps

The term DevSecOps, Development Security Operations, describes a security focused, continuous delivery, software development life cycle (SDLC).

DevSecOps are built upon the general practices of DevOps, adding the security mechanisms that should be followed to secure a CDP. DevSecOps injects active security audits into agile development. And promotes that applying security into the product comes hand in hand with its planning and development rather than to its final version. DevSecOps promotes collaboration rather than handing in security principles to team members, as everyone engaged in the development of the application knows it should be secured. DevSecOps operate on continuous security, as in continuous integration, every change within the code should be checked for its security robustness. DevSecOps operate in layers including: [56]

- Security unit Testing: Testing the security of the applications components is as crucial as testing their functionality.
- SAST (static analysis security testing): Alongside detecting violations in coding best practices, static code analysers detect security vulnerabilities.

Static analysis tools such as SonarQube detects violations in the coding best practices along with detecting security vulnerabilities.

- DAST (dynamic analysis security testing): Unlike SAST, DAST detects vulnerability in the application if communicating externally in its running state. Such mechanism is helpful in detecting violations in servers that have access to the pipeline.

Introducing DevSecOps is crucial when planning and implementing a CDP, as it offers protection for the application from both internal and external threats.

6. CONCLUSION

Working in the software engineering domain does not only mean working with a set of codes. Such requires a methodology and a set of practices that would serve delivering these codes optimally. There are numerous approaches used in software engineering, and continuously integrating (CI) the codes is one of the most widely used ones nowadays. This study tries to mainly answer what are the benefits and possible challenges when adopting continuous integration. As discussed earlier, the stages are not necessarily identical in all continuous integration pipelines but are the most common when corporations adopt CI. Building, thoroughly testing, and keeping the product in a deployable state are essentials.

There are a lot of issues that would come across within these stages, however, the benefits are numerous. Continuously integrating the changes of the code help us track bugs early on, free personal workstations of all the versions of these changes, checking the quality of the code changes, and having the product kept running so that it could be deployed whenever needed and more. All these benefits result in a higher productivity by developers, less frustration, and keeping on track with the deadlines put by the customer. And on the customer side, more satisfaction.

Nevertheless, reaching all these benefits would not come easily, setting up a continuous integration and delivery pipeline (CDP) requires experienced testers and DevOps engineers and that would come hand in hand with higher cost, resource demand and time needed. Besides automation testing problems, which are the most widely reported by corporations when already adopting CI/ CD. However, such drawbacks can be managed and mitigated. Setting up the tools and environments might be initially costly and time consuming, Nevertheless, worthwhile in the long run.

According to the findings based on the discussion on the effort needed for implementing CI, resources and organizational factors are the main challenges. And these could be mitigated by allocating enough tooling and resources, proper training, communication between team members, and providing expertise to team

members whenever needed. Solving integration related problems could be resolved by rejecting bad commits, monitoring build length, and practicing no branches principle. Resolving test automation related challenges, which is the most reported challenge, could be done with dealing with flaky tests and dissecting a complex software to small stand-alone sections. Resolving security issues does not only come with installing strong security mechanisms but also with properly assigning access rules to members and raising awareness on security issues. Following the principles of DevSecOps could be part of a resolution.

Overall, this study highlights the fact that the benefits of adopting CI/ CD outweighs its challenges if there are resources, security mechanisms and proper expertise given to maintaining a CDP. Organizations are to make specific analyses to determine whether to adopt CI/ CD.

REFERENCES

[1] Lucy Ellen Lwakatare, Pasi Kuvaja, Markku Oivo Relationship of DevOps to Agile, Lean and Continuous Deployment, pages 399-415, In Int. Conf on Product-Focused Software Process Improvement, 2016.

[2] Rathnayaka, Kumara, A Review of Software Development Methodologies in Software Engineering, pages 36-48, 2020.

[3] Crispin, Agile testing: A practical guide for testers and agile teams. Reading, Addison-Wesley Professional, , pages 13-18, 2008

[4] J Koivuniemi, Shortening feedback time in continuous integration environment in large-scale embedded software development with test selection, pages 16-18, University of Oulu repository, 2017

[5] BogoToBogo DevOps: phases of continuous integration, https://www.bogotobogo.com/DevOps/Continuous_Integration_Phases.php

[6] Isaac Sacolick, What is CI/CD? Continuous integration and continuous delivery explained | InfoWorld website, https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html, 2020.

[7] Isaac Kapelonis, What is CI/CD? Continuous integration and continuous delivery explained | thenewstack official webpage, https://thenewstack.io/understanding-the-difference-between-ci-and-cd/, 2018.

[8] Brent Laster, Continuous Integration Versus Continuous Delivery Versus Continuous Deployment, O'Reilly Media, Inc., pages 22-34, 2017

[9] What is CI/CD? Continuous integration and continuous delivery explained, https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

[10] Eero Laukkanen, Juha Itkonen, and Casper Lassemius, Problems, causes, and solutions when adoption continuous delivery a systematic literature review, pages 55-79, Aalto University Research information portal, 2017

[11] Mojtaba Shahin, Muhammad Ali Babar, Mansooreh Zahedi, and Liming Zhu, Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges, In Int Con. ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2017

[12] Karl Carenas, , Difference between CI/ CD | Medium webpage, https://medium.com/what-is-ci-cd-pipeline

[13] Version control and continuous integration, tutorialspoint webpage | https://www.tutorialspoint.com/continuous_integration/continuous_integration_version_control.htm

[14] Why git | Atlassian webpage. https://www.atlassian.com/git/tutorials/why-git

[15] Travis CI, build stages | Travis-ci official documentation https://docs.travis-ci.com/user/build-stages/

[16] Pushing and committing changes to git | Github official documentation https://docs.github.com/en/free-pro-team@latest/github/using-git/pushing-commits-to-a-remote-repository

[17] Working with Gerrit, An example | Gerrit official documentation https://gerrit-review.googlesource.com/Documentation/intro-gerrit-walkthrough.html

[18] Saurabh. What is Jenkins | edukera webpage https://www.edureka.co/blog/what-is-jenkins/, 2020

[19] Samuel Brown. Stages of Continuous Delivery. Oteemo webpage | stages of continuous delivery, the build stage https://oteemo.com/2017/11/02/stages-continuous-delivery-part-1-build/. 2017.

[20] Richard Bellairs. What is code quality and how to improve Code Quality. perforce webpage | https://www.per-force.com/blog/sca/what-code-quality-and-how-improve-code-quality, 2019.

[21] Code Quality | SonarQube official Documentation https://SonarQube.org/documentation

[22] Archana Choudary. Integrating testing | Edureka weppage. https://www.edureka.co/blog/what-is-integration-testing-a-simple-guide-on-how-to-perform-integration-testing/Acceptance testing, 2019

[23] Martin Fowler, Continuous Integration | Martin Fowler articles https://www.martinfowler.com/articles/continuousIntegration.html

[24] Steve Neely, S.Stolt. Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not. That Easy, ResearchGate, in Int Conf. Agile, 2013

[25] Daniel Stahl, Jan Bosch. Experienced benefits of continuous integration in industry software Software Product Development: A Case Study. 2013

[26] Michael Hilton, Timothy Tunnel, Kai Huang, Darko Marinov, and Danny Dig. Usage, Costs, and Benefits of Continuous Integrationin Open-Source Projects. In Int Conf,the 31st IEEE/ACM International Conference. 2016.

[27] Haverilla Severi, Impacts of Continuous Delivery in Software Projects Severi Haverila. Aaltodoc. pages 38-46

[28] Mojtaba Shahin, Muhammad Ali Babar, Mansooreh Zahedi, and Limiting Zhu , Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges, in Int Conf 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2017.

[29] Martin Fowler.MartinFowler original continuous integration, the benefits of continuous integration https://martinfowler.com/articles/continuousIntegration.html, , 2016.

[30] Ian Buchanan, Why agile isn't agile without continuous delivery | Atlassian webpage https://www.atlassian.com/continuous-delivery/principles/why-agile-development-needs-continuous-delivery.

[31] What is Scrum | Scrum official documentation https://www.scrum.org/resources/what-is-scrum]

[32] GIT vs SVN | perforce webpage https://www.perforce.com/blog/vcs/git-vs-svn-what-difference, 2018

[33] Garbage collection in Git | Atlassian webpage https://www.atlassian.com/git/tutorials/git-gc

[34] SonarQube Metrics Definitions | SonarQube official Documentation. https://docs.sonarqube.org/7.1/MetricDefinitions.html

[35] Chris ward. Exoscale webpage | What is continuous integration https://www.exoscale.com/syslog/what-is-continuous-integration, 2018

[36] SonarQube Rules | SonarQube official documentation. https://docs.sonarqube.org/7.4/user-guide/rules/

[37] SonarQube Rules | SonarQube official documentation. https://docs.sonarqube.org/latest/user-guide/issues/].

[38] Acceptance testing. Infoq webpage | https://www.infoq.com/news/2017/04/acceptance-testing-delivery/

[39] Quality Assurance in Continuous integration West webpage | https://www.west.com/blog/interactive-services/continuous-integration-in-qa/

[40] Docker containers | Docker official Documentation https://www.docker.com/resources/what-container

[41] Robert Wenner, Extreme Programming and Agile Methods - XP/Agile Universe 2003, in Conf.Third XP and Second Agile Universe Conference, 2003, pages 12-16.

[42] Connect the dots of DevOps value, Charles Betz. How To Sell The Value Of DevOps And Continuous Delivery. Business Case: The Modern Technology Operations Playbook | Forrester report. 2019.

[43] Farah Hariri, Lamyaa Eloussi, and Darko Marinov. An Empirical Analysis of Flaky Tests. In Conf: the 22nd ACM SIGSOFT International Symposium, 2014

[44] Christina Paule, Thomas F. Düllmann, André Van Hoorn. Vulnerabilities in Continuous Delivery Pipelines? A Case Study. In Conf: IEEE International Conference on Software Architecture Companion (ICSA-C), 2019.

[45] Unit Tests vs. Integration Tests. Educba | https://www.educba.com/unit-test-vs-integration-test/

[46] What is continuous integration testing. Educba | https://www.edureka.co /blog/what-is-integration-testing-a-simple-guide-on-how-to-perform-integration-testing

[47] Jez Humble, David Farley. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional .2010. pages: 60-82.

[48] Binary Artificat Repository. Jfrog Documentation | http://help.collab.net/index.jsp?topic=/teamforge178/faq/binary_artifact_repository_overview.html

[49] 8 Reasons for DevOps to use a binary repository manager. Jfrog Documentation | https://jfrog.com/whitepaper/devops-8-reasons-for-devops-to-use-a-binary-repository-manager/

[50] System Testing. Geeks for Geeks web page | https://www.geeksforgeeks.org/system-testing/

[51] Martin Fowler. Martin Fowler articles | https://www.martinfowler.com/articles/continuousIntegration.html#TestInACloneOfTheProductionEnvironment

[52] Continuous testing ensures continuous delivery. GetZephyr web page | https://www.getzephyr.com/insights/continuous-testing-cloud-ensures-continuous-delivery

[53] What is Paas. Microsoft Azure official documentation | https://azure.microsoft.com/en-us/overview/what-is-paas/

[54] Faheem UllahAdam Johannes Raft, Mojtaba, Muhammed Ali Babar, Security Support in Continuous Deployment Pipeline. Conf 12th International Conference on Evaluation of Novel Approaches to Software. 2017

[55] Paul Rimba, L. Zhu, and S. Reeves. Composing Patterns to Construct Secure Systems. Conf: 11th European Dependable Computing Conference.pp. 2015. 213–224.

[56] DevSecOps. Atlassian Web page | https://www.atlassian.com/continuous-delivery/principles/devsecops