ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)

στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

“Survey: Continuous Integration & Testing”

ΘΕΟΔΩΡΟΣ ΚΑΡΑΓΙΑΝΝΗΣ

Α.Μ: MM4160007

ΑΘΗΝΑ, ΙΟΥΛΙΟΣ 2018

Περίληψη

Εισαγωγή: Στην εποχή μας η εμπλοκή του λογισμικού στις ζωές μας γίνεται όλο και βαθύτερη καθημερινά, γεγονός που οδηγεί στην ανάγκη ταχύτερης απελευθέρωσης των προϊόντων στην αγορά. Με δεδομένο τον έντονο ανταγωνισμό μεταξύ των εταιρειών, η πρόκληση που αντιμετωπίζουν όλες οι εταιρείες σε ένα ταχέως μεταβαλλόμενο επιχειρηματικό περιβάλλον είναι να διατηρήσουν και, ει δυνατόν, να διευρύνουν το μερίδιο αγοράς τους. Για την αντιμετώπιση αυτής της πρόκλησης, την τελευταία δεκαετία οι ερευνητές πρότειναν διάφορες τεχνικές για την αυτοματοποίηση της σύνταξης, της κατασκευής και της δοκιμής του λογισμικού. Επιπλέον, σε όλη τη σύντομη ιστορία της, η ανάπτυξη λογισμικού χαρακτηρίζεται από εμπόδια κατά τη διάρκεια πραγματοποίησης σημαντικών δραστηριοτήτων όπως ο σχεδιασμός, η ανάπτυξη και η υλοποίηση. Τα συστήματα συνεχιζόμενης ενσωμάτωσης (Continuous Integration) αυτοματοποιούν την κατάρτιση, την κατασκευή και τη δοκιμή του λογισμικού και αποτελεί επίσης μια πρακτική που έχει προκύψει με σκοπό την εξάλειψη των ασυνεπειών μεταξύ ανάπτυξης και τελικής παράδοσης του λογισμικού στην αγορά. Η αναζήτηση βιβλιογραφίας για σχετικές μελέτες ανακάλυψε 134 άρθρα στον τομέα της συνεχιζόμενης ολοκλήρωσης, τα οποία εμφανίστηκαν μεταξύ του 2014 και του 2018.

Σκοπός: Καθώς ο τομέας της έρευνας ωριμάζει και ο αριθμός των συναφών εγγράφων αυξάνεται, είναι σημαντικό να εντοπίζονται συστηματικά, να αναλύονται και να ταξινομούνται οι δημοσιεύσεις και να παρέχεται μια επισκόπηση των τάσεων και των εμπειρικών στοιχείων σε αυτόν τον εξειδικευμένο τομέα.

Μέθοδοι: Διεξήγαμε μια μελέτη συστηματικής χαρτογράφησης προκειμένου να παρέχουμε μια επισκόπηση της συνεχιζόμενης ενσωμάτωσης και να προσδιορίσουμε τον τύπο της έρευνας και τα διαθέσιμα αποτελέσματα. Στο πλαίσιο αυτής της μελέτης, θέτουμε οκτώ ερευνητικά ερωτήματα, καθορίζουμε κριτήρια επιλογής και συνθέτουμε τα εμπειρικά στοιχεία σε αυτόν τον τομέα.

Αποτελέσματα: Το σύνολο των μελετών μας περιλαμβάνει ένα σύνολο 111 άρθρων (από τα 134 ανακτηθείσα άρθρα) που δημοσιεύονται στον τομέα της συνεχιζόμενης ολοκλήρωσης μεταξύ 2014 και 2018. Μεταξύ των αποτελεσμάτων είναι τα εξής: (1) κατάλογος των απαιτούμενων για την εφαρμογή της συνεχιζόμενης ολοκλήρωσης, (2) τα οφέλη και τα εμπόδια που εντοπίστηκαν σε αυτόν τον τομέα, (3) οι προκλήσεις και πρακτικές που αναφέρθηκαν, (4) ο κατάλογος των συνεχών εργαλείων ενσωμάτωσης και οι δυνατότητές τους και (5) μια επισκόπηση των τάσεων σε αυτόν τον εξειδικευμένο τομέα.

Επίλογος: Η συνεχής ενσωμάτωση μπορεί πλέον να οριστεί ως μια ιστορία επιτυχίας στην αυτοματοποιημένη μηχανική λογισμικού, προσφέροντας έτσι μια πληθώρα νέων ευκαιριών τόσο για τους ερευνητές όσο και για τις εταιρίες ανάπτυξης λογισμικού.

Abstract

Context: Computer software has become a part of our everyday life, and many actions are connected with it. A key factor for software companies is to have the ability to release their products fast considering the competition that exists. The main challenge is to keep or even increase the market share. To address this challenge, for the past decade researchers have proposed various techniques for automating the repetitive tasks that happen during software development. Continuous integration (CI) systems automate these repetitive tasks which are the compilation, building, and testing of software and also it is a practice which has emerged to eliminate issues between development and deployment. Our literature search for related studies retrieved 134 papers in the area of continuous integration, which have appeared between 2014 and 2018.

Objective: As this research area matures and the number of related papers increases, it is important to systematically identify, analyze, and classify the publications and provide an overview of the trends and empirical evidence in this specialized field.

Methods: We conducted a systematic mapping (SM) review study in order to provide an overview of continuous integration, and identify the amount, the type of research and results available. As part of this study, we pose nine research questions, define selection criteria, and conclude to results.

Results: Our pool of studies includes a set of 111 papers (from the 134 retrieved papers) published in the area of continuous integration between 2014 and 2018. Among our results are the following: (1) a list of what is needed to implement CI, (2) the benefits, tradeoffs and barriers identified in this domain, (3) common challenges and practices reported, (4) a list of continuous integration tools and its capabilities and (5) identified trends related to CI.

Conclusion: CI is rising as a big success story in automated software engineering, thus offering a plethora of new opportunities for both researchers and software companies.

Acknowledgments

This Thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study. It is a pleasure to convey my gratitude to them all in my humble acknowledgment. I am extremely thankful to Marinos Kintis and professor Malevris Nikolaos.

Table of Contents

Table of Images

1. Software Development Models

Continuous Integration is a practice that is closely connected with software development models. Based on this fact and in order to better understand this term, we present a short history of software development models.

Software development process can be described as a set of repetitive steps, which are made in order to develop software. The two main categories of software development models are: traditional models and lean models. It is vital to select the appropriate software development model considering the approach that each of them follows, as the choice is very important and influences the success of the product that will be developed but also influences the software cost and the quality of the product.

Software development methodologies appeared in 1970's [1]. The first model was the waterfall model. Agile started to be broadly used in 2001. Nowadays agile methodologies are broadly used instead of traditional methods due to the many advantages that these methods have. The reasons that companies are moving to agile methodologies are to improve quality, reduce delivery time, cost and also improve the processes when it comes to development team coordination. Traditional approaches are considered as heavy weight methodologies in contrast with agile methods which are lightweight [2]. The main differences between these approaches are: user requirements, cost, development strategy, testing, customer involvement, software quality and scalability.

Helena Olsson and Jan Bosch presented a model named "The stairway to heaven" in their study [3], which shows the evolution of software development processes. The first step is a transition from traditional methods to agile. The next two steps are about continuous integration and adopting continuous deployment. The final stage is about research and development based on data that are produced after customer feedback.

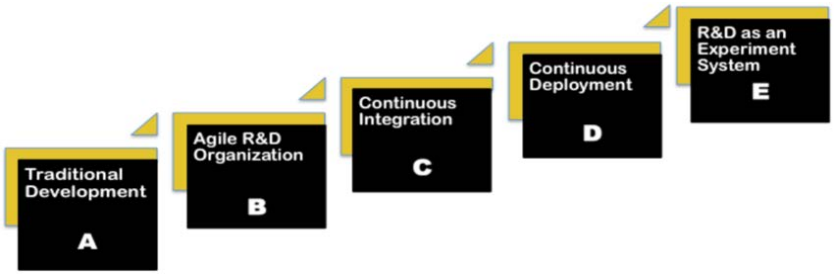


Figure 1. "Stairway to heaven"

A development model should be carefully selected based on the requirements of the software. Software development projects include four common variables [4]:

• Cost is the most important aspect in software development since there is a specific budget that is available.

• Time relates to the schedule that is decided and is connected to the delivery date of the software. If the delivery is late, the only way to deliver on time is to decrease quality or change the scope, neither of which the customer would like to happen.

• Quality is another vital factor for the success of the product. As we mentioned earlier we can deliver working software on time by decreasing quality but the cost will be huge in order to fix the bugs later.

• Features are about the functionality of the project. Developers are responsible to implement the features and it is what they should always focus on. It is very important for the customer and is also a variable that developers have the most control over.

1.1 Traditional Model

Waterfall model is the most popular between traditional software models. The approach that follows is to separate each phase that takes place during the development of the product. Requirements, design, implementation, verification and maintenance are the distinct phases that

exist based on the waterfall model. It is considered as a "top-down" development and each phase should be completed before starting working on the next one.

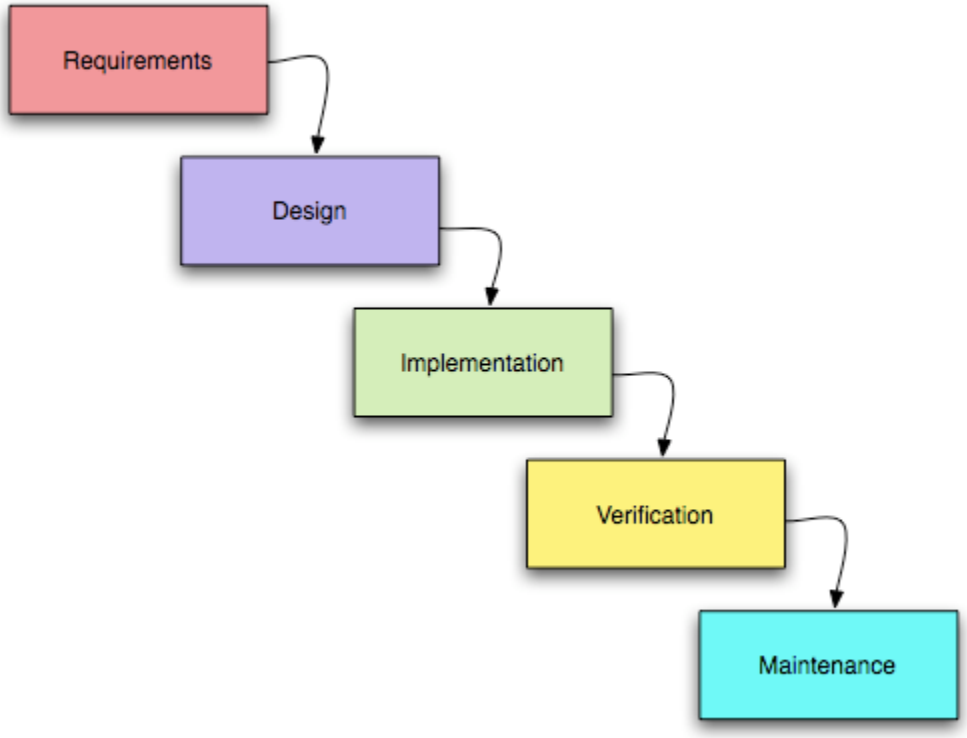


Figure 2. General overview of Waterfall model.

Waterfall model includes five distinct phases which are: Requirements, Design, Implementation, Verification and Maintenance phase (Figure 2). In order to proceed for one phase to the next one, the previous stage must have been completed first. This is because of the fact that the output from a completed stage is used as an input in the next phase. In order to move from one phase to another there is a formal review that is taken place in order for the involved parts (development team, customer) to agree and proceed with the next one[32].

In order to decide that a phase is completed, developers have defined several milestones which have to be met. An advantage of the Waterfall Model is the fact that members of the project who have completed their work on the active phase can take part in other projects. Another benefit of this model is the fact that it fits well in small projects. In addition, based on the fact that requirement and design phases receive are treated as separate phases and are considered as

completed before the implementation phase starts; it helps on saving time and effort, as well as decrease the probability to fail customer expectations.

Regarding disadvantages of the Waterfall model [6], it is not suitable to use it when implementing large projects. It is not so simple to evaluate each phase of the project regarding the time that it is needed and also it is complex to calculate the required budget for each stage. Another disadvantage is the fact that the Waterfall model is not as flexible as agile methodologies are. The testing phase starts after the completion of the development phase and this is something that increases the chance to have more bugs. Returning to previous phases in order to fix the bugs is very difficult.

The Waterfall model is not suitable for projects that do not have clear requirements from the beginning. Project requirements must be defined when the development phase starts, but in reality customers often do not know exactly all the needed requirements from the start. When new requirements are being added after the requirement phase is completed, it causes many problems and decreases the effectiveness of this model. This may result on causing more unwanted issues such as increased cost. Another concern when adopting Waterfall Model is the fact that it is difficult to define the progress of a project. Since each phase is separate from the others, there is not a way to provide the stakeholders with a percentage of project completion [35].

The final recipient of the product does not have the opportunity to become familiar with the system when it is being developed and this is the reason that user feedback cannot be sent back to the developers in order to fix any issues. Based on the fact that the final product is not available until the end of the process, stakeholders take part only during the gathering of requirements phase and after the end of the implementation phase. Quality checks are a big challenge when adopting the Waterfall model because the stakeholders cannot ensure the quality of the product that is being developed before the end of development process.

Except from all the above disadvantages that we analyzed, this model remains one of the most popular models in the software development area suitable for small projects with well defined requirements that does not change during the development lifecycle.

1.2 Agile Methodologies

The mission of agile methodologies is to provide working software after the end of every iteration. Risk is one of the most basic problems in software development. Agile methodologies seek to minimize risk by controlling the four variables of software development [4].

Having in mind these four variables, agile methods try to minimize risk, by giving power to the developers so that they can control as many of the variables as possible and especially the scope of the project. Agile methods are very flexible and are can respond quickly to changes that happen during the development of a product. Basic principles of Agile are shown on Figure 3.

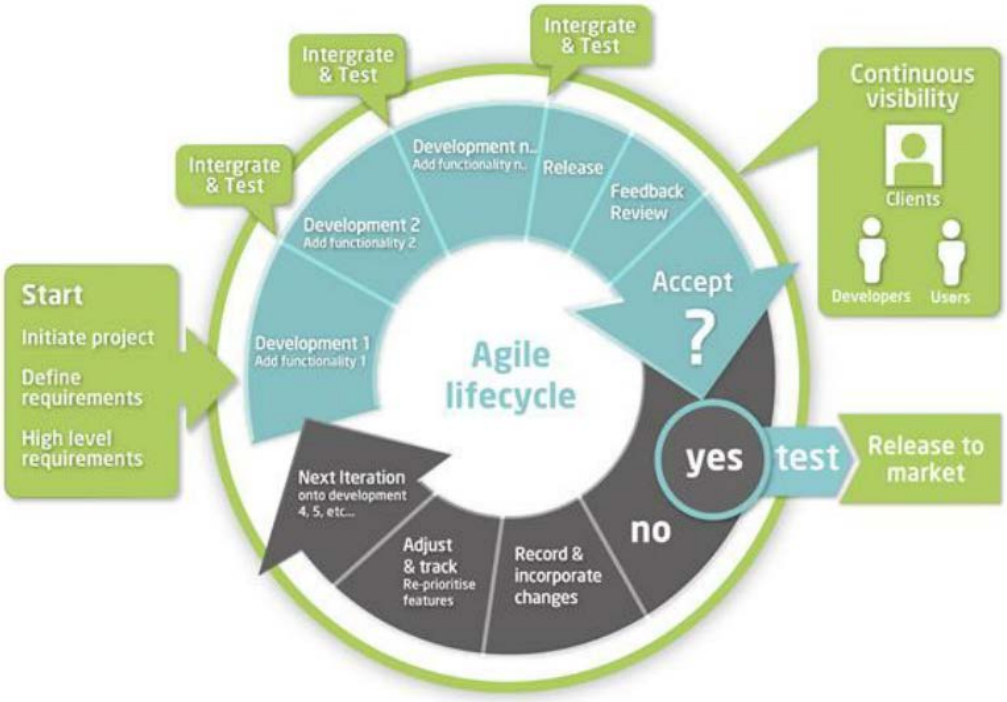


Figure 3. Typical view of Agile Model.

(source URL: https://www.researchgate.net/figure/Agile-Development_fig9_268155804)

Agile development has a main goal which is to provide the customer with working software. The development team should spend most of their time writing code and tests instead of documentation. Due to these facts, agile methods are described as lightweight methods.

Lightweight methodologies like agile, have some common characteristics: they adopt test driven development (TDD) which is about writing tests before code, they provide the ability for frequent product releases, the customer is heavily involved during the development phase and finally they empower maintenance related actions such as code refactoring.

Another fact is that lightweight methodologies have been successfully used in any kind of project (from small till very large). Due to iterations approach, a very large project can be handled as many small projects that can integrate step by step till the final product is implemented and finalized at the end. Organizations are realizing the benefits by adopting agile [8], [9].

Reasons for Adopting Agile [7]

1. Accelerate software delivery
2. Enhance ability to manage changing priorities
3. Increase productivity
4. Improve business/IT alignment
5. Enhance software quality
6. Enhance delivery predictability
7. Improve project visibility
8. Reduce project risk
9. Improve team morale
10. Reduce project cost

Engineering Practices Employed [7]

1. Unit testing
2. Coding standards
3. Continuous integration
4. Refactoring
5. Continuous deployment
6. Pair programming
7. Test-driven development (TDD)
8. Collective code ownership
9. Sustainable pace
10. Automated acceptance testing

The term of Agile has emerged during a meeting of seventeen software developers, who named themselves as "The Agile Alliance". "Manifesto for Agile Software Development" was published in 2001 [10]. The need behind agile methodologies was to formulate a lightweight software development model in order to gain the benefits that the Waterfall Model didn't have due to the fact that it was considered as a heavyweight method. Most agile methodologies aim to minimize risk and provide working software aiming to have short iterations (1-2 weeks).

Each iteration is considered as a small project and includes all the software development phases such as: planning, requirements analysis, design, coding and testing. A single iteration doesn't mean that it will be enough to release a new version of the product but it means that the software that contains the new functionality is ready for release at the end of this iteration. After the end of an iteration, there are several ceremonies in which the team performs a reassessment of development requirements and processes. Agile methods empower direct communication and

this is the reason that agile teams are placed in the same office. The role of a product owner also exists when using agile and this role can be obtained by a customer or a business analyst, who defines product requirements. This role can be performed also by a project manager. Furthermore, each Agile team is consisted by several roles e.g. SET and manual testers, UI/UX designers, delivery leads, back-end / front-end developers. The main values regarding Agile methodologies are described in the Agile manifesto (Figure 4).

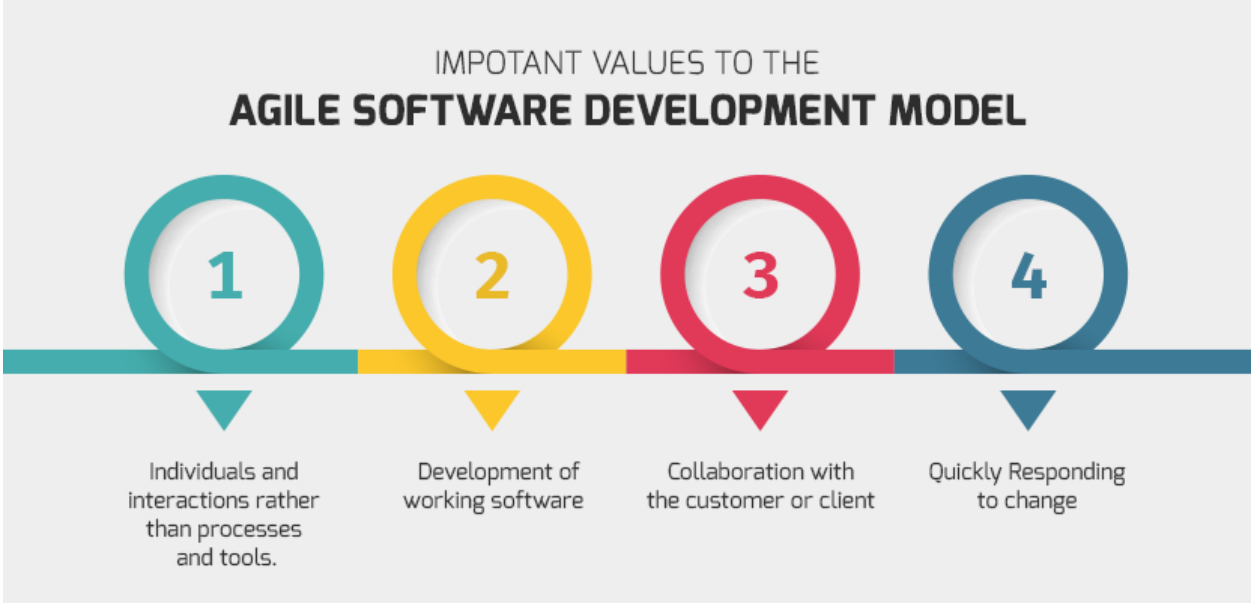


Figure 4. Values of Agile manifesto.

(source URL: https://www.quickscrum.com/Article/ArticleDetails/3041/3/What-is-Agile-software-development-model)

Individuals and interactions rather than processes and tools

Communication and collaboration between developers is the main factor that leads to success.

Development of working software

Creating working software is the main aim of development process rather providing large scale documentation.

Collaboration with the customer or client

Involving customer to the development process is very important in order to ensure that customer expectations are met.

Quickly responding to change

Not all requirements are known in the beginning of the development phase. New requirements are being added to a project during development process and it is important to have the ability to respond quickly on such changes.

1.2.1 Scrum

Scrum is an Agile method [12] and its goal is to help increase the quality during a software development process. Scrum includes events that are called “Scrum Ceremonies” and aim to improve communication between members of the project, provide transparency regarding requirements in order to provide working software. The most important ceremony of this method is “Sprint Planning” in which a plan is agreed and it cannot be changed until the end of the sprint (iteration that normally is between 1-2 weeks). This approach was described first by Hirotaka Takeuchi and Ikudziro Nonaka in 1986 [13].

A Scrum team usually contains 7-8 people who have roles such as developers (Back-End/Front-End), testers (manual/SETs), a business analyst, a product owner and a Scrum Master [14].

The Scrum Master has a key role due to the fact that he/she is responsible for the delivery of the work that it has been planned and also he/she is the connection between stakeholders and the team. Among his/her responsibilities is to “protect” and help the team to focus on their work by removing all obstacles. It is very similar to the Project Manager role but usually this role is known as Delivery Lead nowadays.

According to The Scrum Guide, development team has the following duties [14]:

- Responsible for estimations on tickets that are part of the sprint backlog.

- Take technical decisions.
- Be self-organized.
- Act as one team and not individually.
- Acquire all necessary skills for a product release.
- Developing software and show it to the customer when it is finished.
- Personal development strategy.
- Present the work to the Product Owner.

1.2.2 Extreme Programming (XP)

XP is considered as a lightweight agile methodology [15] and some of its main characteristics are the following:

• **Heavy customer involvement:** A customer representative participates during sprint planning ceremonies and he/she is also involved possibly when writing acceptance tests.

• **Test Driven Development:** It is about writing tests before code. In the beginning the tests are failing and when the implementation is completed then the tests have to pass.

• **Pair programming:** It is a practice during which 2 developers are working together. The main benefit is sharing knowledge which is valuable in order for new members to be familiar with the project or other similar occasions, despite the fact that the development speed is decreased.

• **Short iteration cycles and frequent releases:** Release cycles are divided into small iterations which last 3-4 weeks and releases are frequent e.g. every 4-5 weeks. Quick feedback is a characteristic that describes this method since a customer representative is involved and the release cycles are small. Constant and often integrations are happening after the implementation of every feature is finished. This is something that helps ensure that nothing broke and improve the quality of the software.

XP is characterized by the following key activities [4]:

• **Designing:** Design is a key factor and the base that the product will be built on. Architecture decisions and the choice of the technology stack are very important factors and influence the rest of the development phase but also the success of the final product.

• **Coding:** Coding is about actual implementation. In XP the code quality and the code health is very important aspect and it is a main difference comparing with the philosophy of traditional models when it comes to code. Clean code (self documented) and coding standards are some of the characteristics that show the value of the code since it provides the required functionality to the product.

• **Testing:** Test-driven development (TDD) is a practice that is closely connected with XP. The approach that describes TDD is to write tests before the implementing the functionality. A complete functionality contains tests that pass and usually it is a signal that shows that the code works as expected.

• **Communication:** Communication between team members is an important aspect. The customer has the ability to share the knowledge of the business with the development team. The developers have the required technical knowledge about the implementation. Acting as a team and communicating in a good way is important and helps to develop quality products.

1.2.3 DevOps

DevOps is closely connected to continuous integration. Testing, integration, delivery, and deployment are considered as continuous processes in the context of DevOps. In addition to these, DevOps is an approach that helps to orchestrate all the parts that are related to the development lifecycle of products. Stakeholders, development teams, operations, and other departments cooperate to continuously deliver software that helps the business to achieve its goals.

The term DevOps is about "Development" and "Operations", which are two traditional departments in software companies. Development department is responsible to implement working software. Operations are responsible to ensure stability of all the integrated parts of the software. DevOps aims to fill the gap between development and operations and improve the collaboration between them. Another aspect is that DevOps maintain shared processes and tools in the company. Despite the fact that DevOps is about operations, tools etc. it adopts agile culture that puts people above tools and processes.

2. Software Testing

Success of any software product is determined by the quality of that software. This gives software quality assurance a great opportunity in software industry and customer satisfaction drives it. To develop a product of good quality and without any defects within the cost and time constraints have become critical. Implementing such products, with minimum or no bugs is a difficult task. This is the reason that the concept of software testing has got its existence. It also provides final evaluation of other activities such as requirements specification, software design, and coding.

As it is stated in the following research paper [34] "software testing is an activity, which is performed to evaluate correctness and functionality of software for assuring fulfillment of user requirements and expected quality. IEEE defines software testing as the process to evaluate the system or its components manually or by automated means to determine whether it fulfills the user requirements or to find the difference among actual result and expected result. Hence, the software testing is to execute software to identify defects or any missing features that were expected by the user requirements. Software testing results in improved quality and effectiveness of the software system, if it is executed appropriately. Detecting the defects in software and removing those defects before the release of software leads to reduced maintenance cost".

Improved software quality is one of the major reasons to perform testing. Software quality is improved by ensuring that the software product works as expected based on the requirements that the customer has defined.

Quality is defined by the following factors: reliability, portability, efficiency, security, usability, correctness, maintainability, compatibility etc. During the testing process all the quality factors should be checked and ensured. The main goal of the testing process is to detect errors or defects. The result of this is to improve the efficiency of the software and the reliability but also the other factors that we already mentioned previously. Software testing can be divided into two types: automated testing and manual testing.

2.1 Manual Software Testing

Manual testing is a type of testing that is being done by humans. Manual testers generate test cases in order to interact with the software and trigger specific functionality or journeys that they would like to test. Test cases are scenarios which describe the expected behavior that the software under test should have, and are written in natural language. It is a time consuming process since everything should be done manually [17]. Manual testing is useful process and it should not be completely replaced from automated testing.

Manual testing has the following disadvantages:

- Time consuming process
- Requires more testers as testing cannot be done in parallel
- Possibility of human error
- Lack of reusability (In the case that there aren't any common journeys/functionality)

2.2 Automated Software Testing

The second type of software testing is the Automated software testing. This kind of testing helps to be more efficient when it comes to quality assurance. Human involvement is not heavily needed comparing with manual testing and after writing the tests for the first time then human effort is needed only to maintain them. Selecting the testing tool and setting up the testing environment is an important step before the creation of the tests.

The goal of automating software testing is to minimize human effort and also to reduce time and cost due to high reusability of testing scenarios but also reduce the maintenance cost of the software platform.

Automated testing has the following benefits:

- Reusability
- Less time and cost
- Supports testing in parallel
- Human effort is minimized

Automated testing has the following drawbacks:

- Need a testing environment which is properly set up.
- Software Engineers in Test are needed to write tests and to maintain the testing platform.
- Manual testing cannot be fully discarded

How often and when should we run automated tests? The answer is clear, as often as possible and as quickly as possible. Despite the fact that the answer is correct it is not complete though. Continuous Integration is the process that provides the ability to run automated tests often and quickly.

3. Continuous Software Engineering

Continuous software engineering is an important area that we will analyze in this study. It is about development, testing and deployment of software but also has other aspects such as getting quick feedback. Continuous software engineering involves three phases: Business Strategy and Planning, Development and Operations. There are many initiatives that are named as 'Continuous.' Continuous integration is a practice that has been evolved during the last 4 years but there are also other methods such as continuous deployment which are not widely adopted yet [19].

Business Strategy and Planning

Planning is about actions and forecasts. Forecasts can relate to trends but actions rely on defining processes, resources and ways to achieve company's goals.

Continuous planning involves implementing planning practices continuously, not just as part of a top-down annual event. Planning is a never ending process that has the ability to respond to changes in a business environment. Regarding software development, continuous planning refers to the capacity to plan in rapid parallel cycles.

Continuous Planning involves business stakeholders but also members of the development team in order for everyone to be aligned and respond to changes in an efficient way.

Continuous planning in an agile environment is being done in small increments and with more people comparing with waterfall methods. The plan is for the next iteration which lasts normally around 2 weeks time. Continuous planning has many aspects and is not closely connected with the development team but includes also higher level planning.

Development

Continuous integration (CI) is the most popular among Continuous terms. Extreme Programming (XP) method is closely connected with CI and this is a reason that it is so popular practice. It is not very clear though what other practices are considered as part of CI. We can define CI as a process which helps us to trigger repetitive tasks such as compiling, running tests, verifying quality and deploying the developed software. Feedback is a benefit that CI provides to the development team in order to quickly identify issues and fix bugs.

Continuous integration has many advantages as a practice. It helps to improve communication and thus productivity also during the development phase. In addition, CI helps to orchestrate development and operations so this is the reason that CI is closely related to DevOps. Continuous deployment and continuous delivery as very similar continuous terms with CI and in order to apply those CI is a prerequisite.

Continuous delivery (CDE) has been defined as the ability to release software when it is decided. In reality, this means that a new version is ready to be delivered as soon as the implementation of a feature is finished. This ability is very crucial in software communities.

The adoption of CDE has many benefits. It is important for the business value to be delivered quickly to the customers so that the product to be available to the users. Users can give quick feedback to the development team. Both are crucial factors in order to implement the right product which will return the investment to the company. Without CDE the customer has to wait till a major release in order to receive feedback and decide for the upcoming work that should be done. Productivity and efficiency are also two important improvements.

CDE practice adopts all the benefits that frequent releases have. The new features can be tested more easily and quick due to the fact that there aren’t many changes in the code. Finding and fixing bugs is a much easier job. CDE pipeline can also automatically roll back a release if there are major issues detected. Risks are reduced and as a result product quality can be improved significantly.

Continuous deployment (CD) practice is the next step of CDE since it automatically and continuously deploys the software to production. The main difference between continuous deployment and continuous delivery is that continuous deployment practice is about automatically deploy the final product in a production environment. Continuous delivery is a prerequisite in order to adopt continuous deployment. CD uses a deployment pipeline which helps to deploy the software without any manual process included during this final step.

Software companies constantly try to deliver value to their customers. Agile methods have helped to increase flexibility during software development phase. The highest priority is to satisfy the customer through continuous delivery of quality software and in general deliver working software frequently. CD focuses on software functionality to be continuously deployed to the production environment, where end user feedback is the source of data in order to drive innovation [18].

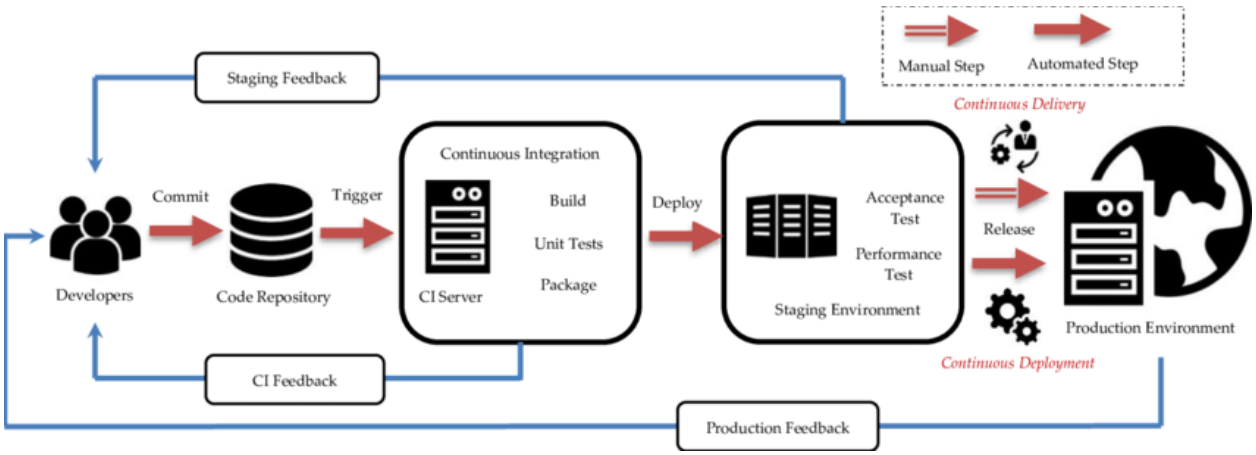


Figure 5. The relationship between CI, CDE, CD.

(source URL: https://www.quickscrum.com/Article/ArticleDetails/3041/3/What-is-Agile-software-development-model)

Continuous verification is a practice which is about continuously testing the software that is being developed and not handle the testing phase as a separate phase. Waterfall model considers verification and quality as separate activities which can start only after requirements, design and implementation are completed. Continuous testing seeks to integrate testing activities as closely

as possible with coding in order for bugs to be fixed quickly. Prototyping is a practice in order to tackle early verification of requirements and it is broadly used during user interface and user experience design phase.
Results of surveys show that continuous testing reduces the overall development time almost 15%. This fact suggests that continuous testing can be an effective tool to reduce one of the types of wastes, namely that of waiting time.

Software development seeks to satisfy regulation standards on a continuous basis, rather than adopting an approach to ensure compliance only when a major issue is found. This process is called **Continuous compliance**.

Continuous security is about handling security as a key concern during all phases of the development lifecycle and even post deployment and not just a nice to have feature of the software.

Operations

Continuous use is a step further from the initial adoption of software due to the increased return of investment that is noticed on systems that adopt continuous use.

Continuous trust is about keeping the trust that has been established after fulfilling customer expectations and requirements by delivering quality software on time. Initial trust is more important in occasions that a quick transaction is made. Continuous trust is very challenging due to the fact that it is constantly being recalculated by customers and the users of the product. Continuous use is closely connected to continuous trust.

Continuous monitoring is an important process since it gives the ability to detect many problems such as performance issues or quality of service related issues or help to ensure the fulfillment of service level agreements (SLAs).

Improvement and Innovation

Continuous improvement is connected to lean methodologies and it is a practice which is based on collecting data from feedback in order to improve the current state of the product such as user interface redesign etc.

Continuous innovation is a process that is responsive to changes and it is used during the entire lifecycle of planning, development and post deployment. Innovation is the process where new ideas are transformed into features or products which have value for customers. A form of continuous innovation is beta testing, which is closely connected to the software industry. This concept is far more matured nowadays and it is transformed into other techniques such as A/B testing where new features are presented to the users and their response is used to take decisions in order to implement those that add more value to the product.

4. Continuous Integration (CI)

4.1 History

Continuous Integration (CI) was first introduced in 1991 by Grady Booch and he describes it in his book Object-Oriented Analysis and Design with Applications: “At regular intervals, the process of continuous integration yields executable releases that grow in functionality at every release... It is through these milestones that management can measure progress and quality, and hence anticipate, identify, and then actively attach risks on an ongoing basis.” This idea was then one of the foundation approaches of Extreme Programming (XP) in order to help on improving software quality [21].

Martin Fowler published an article in 2000 and it was the beginning of a new era for Continuous Integration [22]. According to Martin Fowler the ability to integrate as many times as it is possible during the software development, but also the ability to support automation in order to achieve all the above is very important. Automating the build process and running the tests after each new commit is pushed in the existing codebase can provide the development team with huge benefits.

Martin Fowler was involved on implementing the first CI server, Cruise Control in 2001. Today there are over 40 different CI systems, with Jenkins to be the most popular along with Travis CI, and Microsoft Team Foundation Server (TFS).

4.2 Overview

Continuous Integration is a software development practice which aims to automate the repetitive steps that are being performed during development of a software product. Nowadays Continuous Integration is a widely adopted technique with many development teams to find that it is an approach which leads to improved software quality due to helping find bugs quickly but also increases team morale and productivity. Continuous Integration is a popular technique used in

software development and it is emerging as one of the biggest success stories in automated software engineering.

A typical CI system continuously monitors the version control system for changes and whenever a change is detected then a build is automatically triggered in order to compile and run the tests. CI is built upon automation approach for compiling, bundling, linking dependencies, packaging into an executable form, and running automated tests. CI server provides early feedback which is very important benefit and useful to the development team in order to identify issues and as a result to produce quality software that can lead to frequent release cycles.

4.3 Research methodology

We discuss next the following aspects of our research method:

- Overview.
- Goal and research questions.
- Article selection.
- Final pool of articles.
- Data extraction.
- Data synthesis.

This study used systematic mapping to identify the status of the research that relates to Continuous Integration and Testing [23]. Compared with other methods, such as traditional literature reviews, a mapping study offers the ability to investigate a wide range of the topic, while sacrificing depth. In the context of this study, a systematic mapping method was more appropriate as it provided a formal and well-structured approach to extract data and synthesize them.

Primary search

The search terms that we used were ‘continuous integration’ and ‘continuous testing’ due to the fact that they considered being the most obvious primary search terms. The chosen terms were used as the search keywords in Google Scholar.

The mapping process

Before starting we agreed the research questions in order to provide a general scope for the study. Based on this scope, along with the primary search terms we tried to find research papers stored in several digital databases. After a collection of papers was found, each paper was analyzed using inclusion criteria in order to identify papers that are aligned with the scope of the study. Those papers were further analyzed to find keywords that could be used to classify the research being conducted in the area. Finally, the classified papers were aggregated, visualized and mapped in order to answer the research questions posed in this study.

Inclusion criteria

Every research paper that we collected had to originate from an academic source, such as a journal or conference, and clearly focus on Continuous Integration in software engineering, based on the primary search terms. Publications that met these criteria were further analyzed, in order to highlight the most relevant ones in the area of Continuous Integration and testing in software engineering.

Research questions

The purpose of this research is to classify current research papers, and identify trends in the literature, which relate directly to Continuous Integration and Testing.

Therefore, the guiding research question of the study is:

“What is the need of implementing Continuous Integration?”

To answer the main research question, eight research questions that relate to various aspects of Continuous Integration were identified. Separating the main research question into smaller and more specific questions enables the topic to be investigated from different perspectives, while

also providing the results needed to answer the main research question. The additional research questions are described below.

RQ1: What is the publication trend to CI in software engineering?

Rationale: The intention of this question is to illustrate the interest in the research area from 2014 to 2018, as well as identifying the primary sources of literature in the field. This study assumes that the publication rate is a sign to understand the research interest in the area of Continuous Integration and testing, while the most prominent sources of research in the field are those journals and conferences that have the highest publication frequency of relevant literature.

RQ2: What type of research has been undertaken in the area of CI?

Rationale: The intention of this question is to highlight the type of research that has been done in the area. By answering this question the study aims to understand the maturity level of the research area, but also check if the field is still maturing and the research is focused on developing methodologies to support future research efforts.

RQ3: What types of contributions have been made in the area of CI?

Rationale: The intention of this question is to highlight the contributions of research that has been done in the area. We illustrate the types of contributions, as well as analyzing these contributions by conference and journal publications.

RQ4: What are the most influential articles in terms of citation count?

Rationale: The intention of this question is to highlight the articles in terms of citation count in order to investigate which of them have the highest influence among software engineers and software engineering in general. Also this research question analyzes the relationship between the citations for each article and its year of publication.

RQ5: What is needed to implement CI?

Rationale: The intention of this question is to highlight what is needed in order to implement CI. By answering this question the study aims to understand and define the important parts that are needed in order to adopt CI.

RQ6: What are the benefits of using CI?

Rationale: The intention of this question is to identify the benefits that CI offers to a company. By answering this question the study aims to understand the importance of CI to the development team and also to the customers.

RQ7: What are the tradeoffs of using CI?

Rationale: The intention of this question is to highlight the types of tradeoffs when using CI. By answering this question the study aims to address these tradeoffs that developers face during implementing or using CI.

RQ8: What are the barriers of adopting CI?

Rationale: The intention of this question is to identify the barriers of adopting CI, as well as understanding the type of barriers and analyzing them. By answering this question the study aims to better understand the obstacles and unmet needs that developers face when adopting CI.

RQ9: What are the developers' needs and motivation of using CI?

Rationale: The intention of this question is to highlight the needs and motivation from the development team perspective. By answering this question the study aims to highlight specific research themes, as well as identifying the areas of CI that need further research in order to address all the needs and establish a CI culture within a company.

5. Results of the survey

5.1 What is the publication trend to CI in software engineering?

We illustrate the year-on-year growth in publications relating to CI in software engineering (Tables 1-5). The first publications identified in this study are from 2014. From 2014 to 2015 the publications in the field were increased by 50%. It should be noted that this increase shows that CI would be a rising success story in the upcoming years. We classify all the research papers that we studied and also we add a description that includes further details for each group.

Classification (CI)	Description	Research Papers
Tools & improvements	CI tools and improving the process of testing (e.g. Regression & Performance testing).	[30], [42], [47]
Challenges, effects, trends, impacts.	CI challenges, trends, impacts and effects (e.g. social effects of CI).	[19], [25], [38], [41], [44]
CI flow & Agile fit	CI flow & the relationship between CI and agile.	[11], [39], [43], [45], [46]

Table 1. CI related publications (2014)

Classification (CI)	Description	Research Papers
CI adoption & Agile	CI adoption using Agile methodologies.	[1], [8], [9], [50], [63], [67]
DevOps	DevOps and bridging gaps between processes.	[49], [53], [55], [64]
Quality & Productivity	Software quality & productivity.	[54], [61], [68]
Improvements & Continuous Deployment (CD)	CI to CD and also propose patterns and methods for CI.	[20], [51], [56], [57], [58], [60], [69]
Containerized CI	Virtual platforms & containerized CI using Docker.	[52], [59], [62], [65], [66]

Table 2. CI related publications (2015)

Classification (CI)	Description	Research Papers

Value, Costs, Benefits, needs.	Usage, costs, benefits and value delivery of CI.	[21], [32], [70], [71], [76]
CI, CD & CDE	CI, CD & CDE related topics.	[31], [72], [73], [80], [82], [83]
Micro-services	Micro-services architecture and its connection with CI.	[74], [78], [79], [81]
Test Prioritization	Test & build prioritization.	[18], [26], [77]
Agile & DevOps	Agile and DevOps related topics.	[16], [71]

Table 3. CI related publications (2016)

We would like to highlight the growth in publications relating to CI in software engineering in the year 2017. From 2014 to 2017 the publications in the field were increased by 75% including many new topics that appeared and were investigated such as Continuous Delivery & Deployment and also CI issues that are based on software that is build on micro-services architecture. Further research has been made regarding build failures, flaky tests and methods in order to avoid such breakages. In addition to these, Jenkins seems to be the most popular CI server, while new improvements are being proposed regarding testing processes and deeper research is made that relates to CI tradeoffs, impacts and impediments.

Classification (CI)	**Description**	**Research Papers**
CD & CDE	CD & CDE approaches, tools, challenges, solutions.	[28], [85], [89]
Build issues	Build failures/breakage and best practices to avoid such occasions.	[33], [84], [86], [92], [96]
Micro-services	Micro-services architecture and its connection with CI along with benchmark requirements.	[82], [93], [103]
Impacts/Impediments	CI tradeoffs, consequences, impediments,	[80], [81], [83], [88], [91]
CI servers	Jenkins CI	[104]
Containerized (Docker/Cloud)	Containerized CI on the cloud and also service based CI.	[102]

Testing process improvements	Static code analysis tools, regression testing, security testing, exploratory testing, TDD.	[35], [87], [90], [97]
Continuous Software Engineering	Continuous terms.	[4], [95]

Table 4. CI related publications (2017)

During the first half of 2018 we found that there are some new topics regarding CI that didn't appear in the previous years. For example many research papers are focused on reporting when using CI, investigating CI costs and how CI influences the developer communities. Google scale advancements and improvements are mentioned, while topics such as Agile, DevOps and Continuous terms family are some areas that gathered the interest of researchers.

Classification (CI)	Description	Research Papers
Reporting	CI reporting tools and methods.	[111]
Influence, cost	Use and misuse of CI, cost effective CI and how CI influences developer communities.	[99], [100]
Google Advances	Google advances that are related to CI and Testing.	[105], [109]
DevOps	DevOps improvements.	[106], [110]
Agile	Agile adoption motives & team performance when using CI.	[7], [107]
Continuous Software Engineering	Continuous terms especially Continuous auditing, experimentation, improvement, monitoring.	[108]

Table 5. CI related publications (2018 1st semester)

5.2 What are the most popular types of research that has been undertaken in the area of CI?

Figure 6 provides a breakdown of the type of research being conducted in the area of CI in software engineering. The majority of research conducted is based on these six topics: Agile

Methodologies, DevOps along with CD/CDE, build issues, CI in depth including needs, benefits, costs, impacts and best practices, Containerized CI using Docker and also CI based on cloud technologies such as AWS and finally many topics regarding methods for various kinds of testing.

The most common type of research is related to CI in depth, which solves a particular problem which is to understand each CI perspective. CI benefits, needs, impacts, costs are some of the common categories that we included in this type of research. The next common type of research is Continuous Delivery (CDE) and Continuous Deployment (CD) along with DevOps practices. Less prominent types of research include evaluation of CI usage on various kinds of testing such as regression, GUI, performance and security testing. Furthermore, less prominent types of research include the examination of the reason for build breakage and the implementation of CI using containerized services in the cloud.

Early research efforts in 2014 possessed a strong focus on software development methods research, especially Agile. The trend in publications relating to CI is closely connected with investigating the various aspects of CI in software engineering ecosystem and collect metrics and data in order to improve testing processes and deliver better value to the customer by implementing in parallel CDE and CD.

The most popular types of research can be classified into six groups.

1. CI-in-depth category includes research that has been done in order to understand CI challenges, trends, impacts, various types of effects (e.g. social effects of CI) etc.
2. Testing category includes research that has been done on improving the testing process via Continuous Integration.
3. DevOps category includes research topics on the collaboration between Development and Operations department and also includes topics that focus on Continuous Delivery and Continuous Deployment.
4. Containerized-CI category includes research that has been done on Cloud solutions and platforms such as Docker.
5. Build-issues category includes all the research that is done on identifying issues that are related with build failures and also propose solutions and best practices to tackle this important topic for CI.

6. Agile-Methods category includes research that has been done on the relationship between agile methodologies and CI practice.

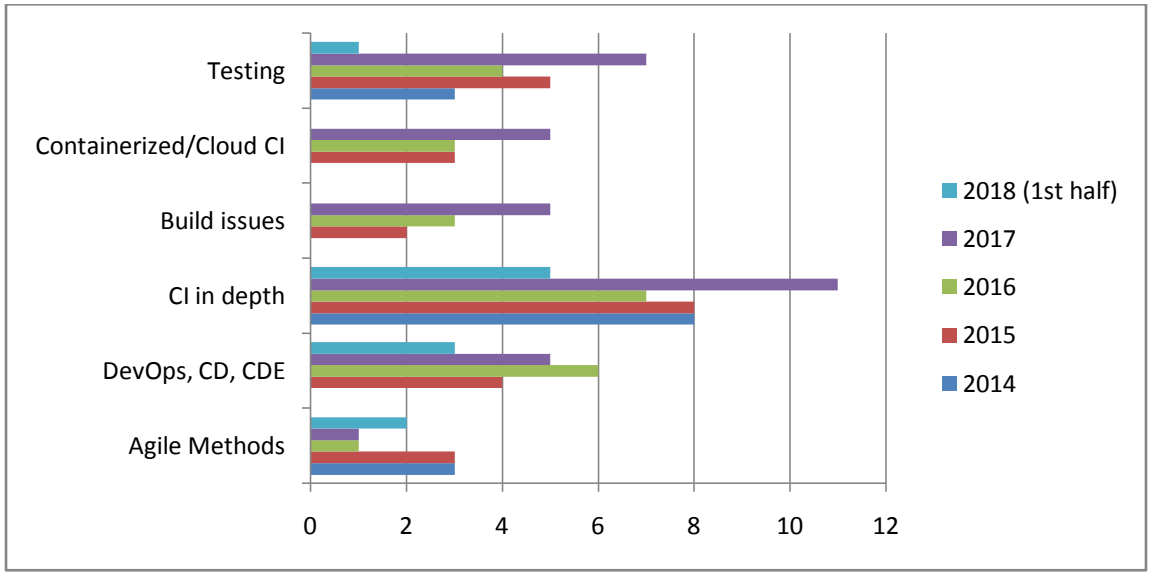
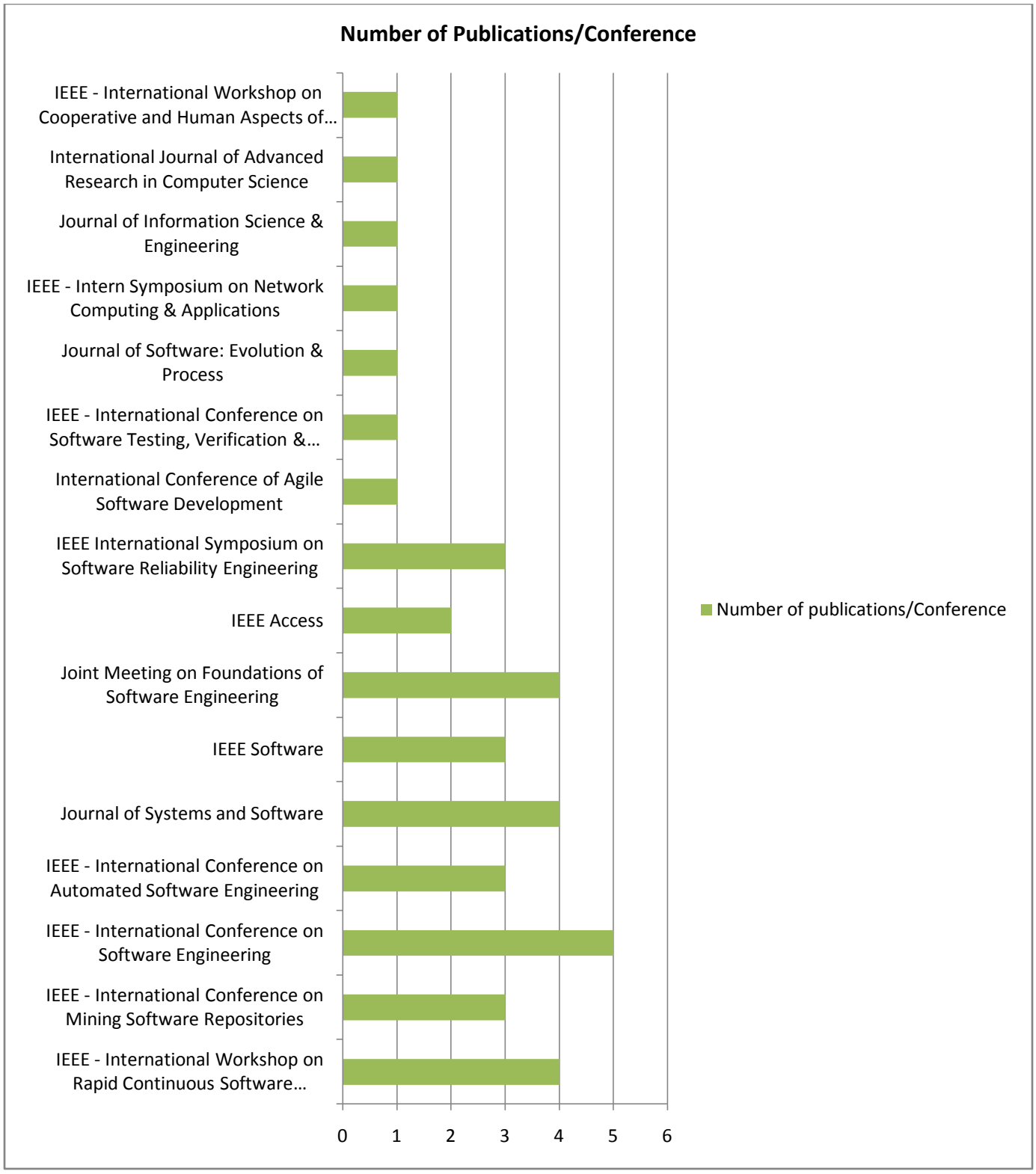


Figure 6. Popular topics: Number of research papers/Year

5.3 What types of contributions have been made in the area of CI?

In the context of this study, we also focused on finding some trends regarding the contributions that have been made in the area of Continuous Integration. We analyze these contributions by conference and journal publications for the period 2014-2018.


Number of Publications/Conference
IEEE - International Workshop on Cooperative and Human Aspects of...
International Journal of Advanced Research in Computer Science
Journal of Information Science & Engineering
IEEE - Intern Symposium on Network Computing & Applications
Journal of Software: Evolution & Process
IEEE - International Conference on Software Testing, Verification &...
International Conference of Agile Software Development
IEEE International Symposium on Software Reliability Engineering
IEEE Access
Joint Meeting on Foundations of Software Engineering
IEEE Software
Journal of Systems and Software
IEEE - International Conference on Automated Software Engineering
IEEE - International Conference on Software Engineering
IEEE - International Conference on Mining Software Repositories
IEEE - International Workshop on Rapid Continuous Software...
0
1
2
3
4
5
6
Number of publications/Conference

5.4 What are the most influential articles in terms of citation count?

This research question analyzes the relationship between the citations for each article and its year of publication. The x-axis is the number of citations, and the y-axis is the title of the article. Fig. 7-10 shows the data that are related to 2014-2017.

The most influential research articles can be classified into four groups.

1. Agile methodologies & CI.
2. Continuous software engineering.
3. Techniques for improving several types of testing in CI.
4. DevOps (Journey to CD & CDE, micro-services architecture).
5. Other CI related topics (e.g. build breakage, costs, benefits, tools).

Figure 7 shows that the most cited article was about modeling continuous integration practice differences in industry software development. This fact shows that there is much interest on find a way to model the differences and the various aspects of a new methodology like CI. Trends and challenges is the second hot trend that we noticed and the third most cited is about testing improvements that can be achieved via CI practices.

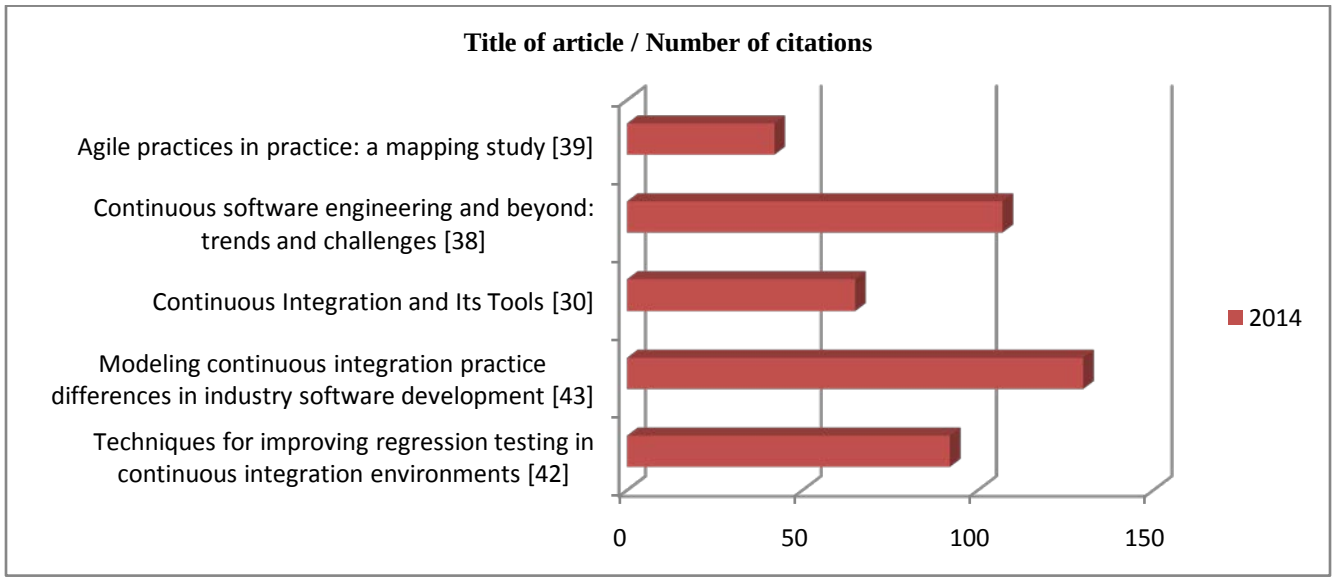


Figure 7. Most influential articles (2014)

Figure 8 shows that the two most cited articles were about the challenges of adopting Continuous Delivery after establishing Continuous Integration and about quality and productivity outcomes in GitHub. This fact shows that after the more general researches that were done in 2014, the interest is shown on some key factors that help on implementing a successful project while adopting CI practices. Considering and improving quality and productivity but also tackling the challenges in order to adopt Continuous Delivery are very popular types of research for this reason. Delivering as soon as possible the value to the customer is one of the most important benefits of CI and CD. Other than these, the results show that DevOps is by far a very interesting research topic where the three next popular articles are focusing and analyzing topics such as inspecting performance benchmarks and understanding the gap on the road to Continuous Delivery.

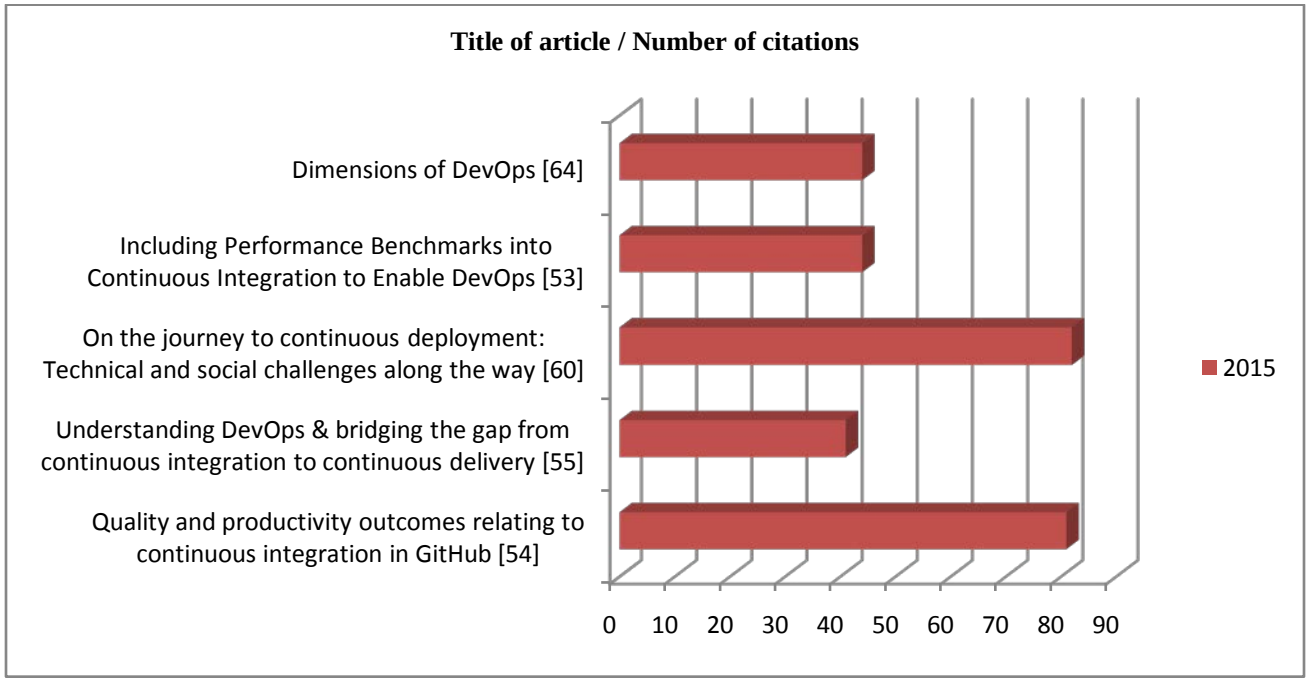


Figure 8. Most influential articles (2015)

Figure 9 shows microservices is a very popular topic of research along with large scale agile transformations. Since agile methodologies are widely known, the results show that the interest is on challenging situations such as large scale transformations. Furthermore the results show

that researchers show a great interest on investigating CI more in depth and analyze the usage, costs and benefits along with analyzing modern release engineering topics and aspects.

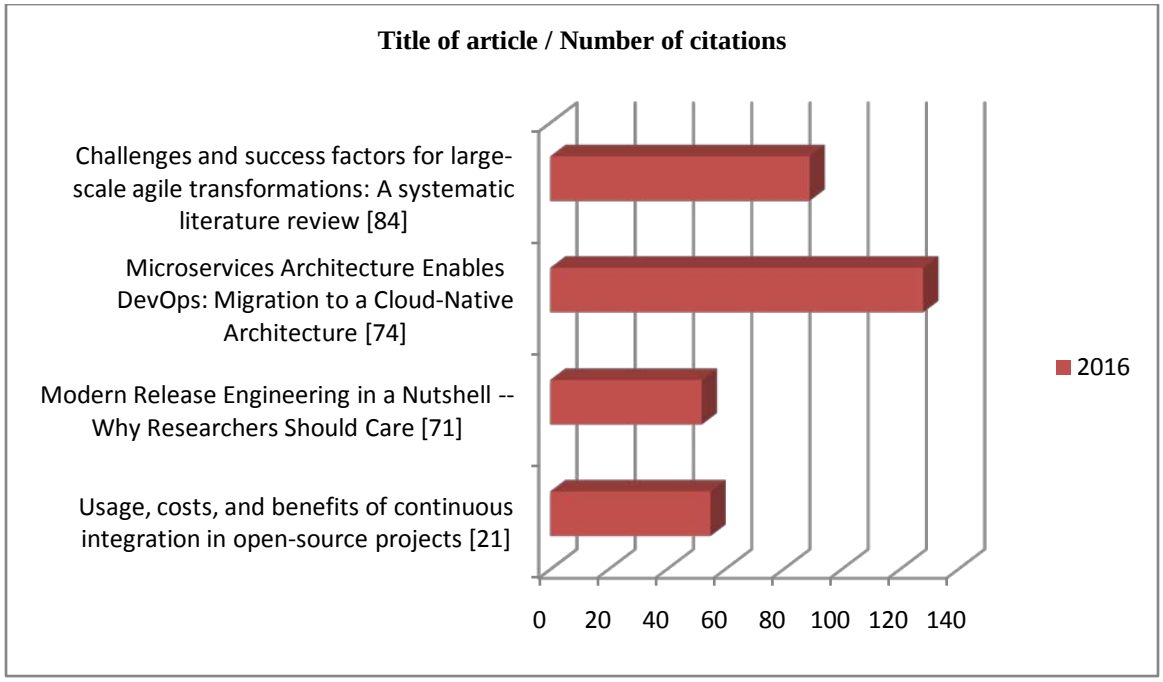


Figure 9. Most influential articles (2016)

Figure 10 shows that the most cited article is about continuous software engineering. In 2017 most of the continuous family terms are widely known and there is big interest for researchers in order to analyze more this modern technique. Continuous Integration, Continuous Delivery, Continuous Deployment are the most popular continuous terms based on the results about most cited research papers but also researches on build breakages reasons and ways to tackle these issues are high cited also.

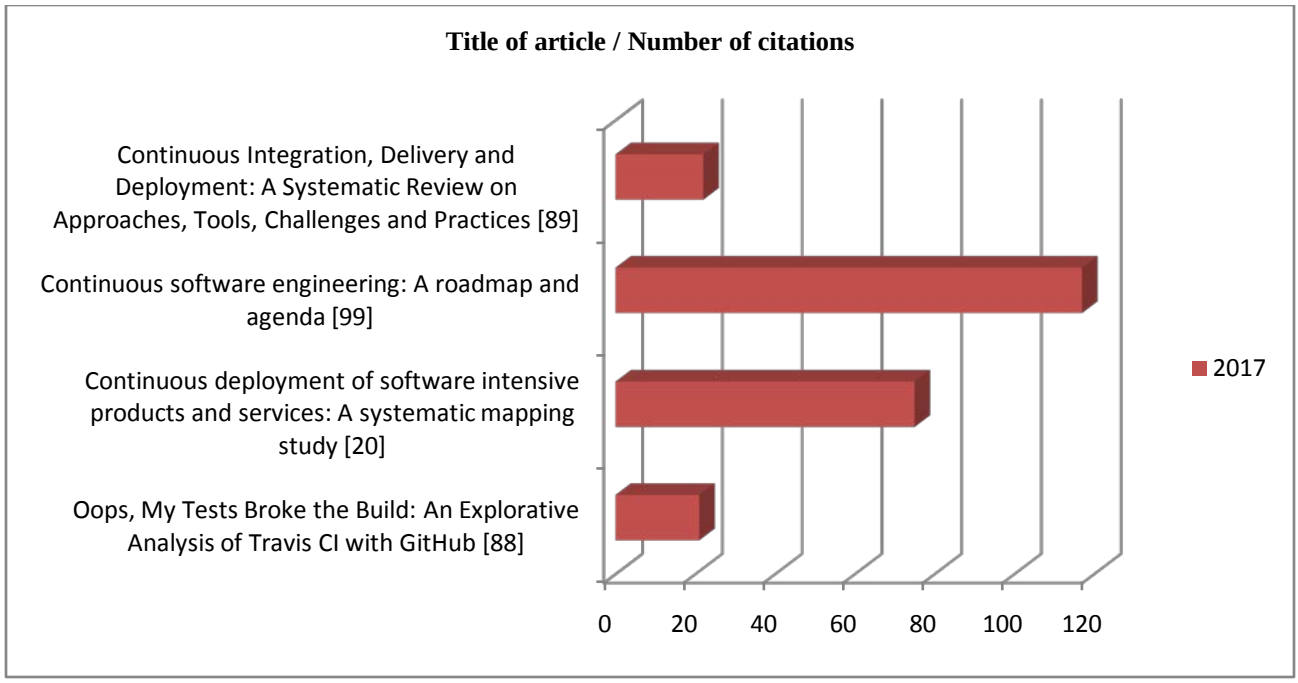


Figure 10. Most influential articles (2017)

Regarding the year 2018 we haven't managed to retrieve any high cited article so far that was related to Continuous Integration due to the fact that we managed to screen research papers that were published in the first semester of the year.

5.5 What is needed to implement CI?

Version Control System

A version control system provides a space that is called repository in order for the developers to have a single point of truth and integrate their code with the rest of the application. There are many strategies regarding how the version control system should be used by the development team.

Some common steps are:

- A new branch is cut by the master branch in order for the developer to work on a new feature or bug.

- This branch should be kept synced with the master branch in order to resolve any conflicts until the implementation is completed.
- After testing is completed, the developer is free to merge his branch into master so that a build can run and ensure that nothing broke.

Some advantages of version control systems are:

- It keeps a history of the changes that are done into files.
- It provides details regarding who made a change when and into which context.
- It gives the ability to rollback any change that is done.

CI-server

A CI-server is a crucial part in order to implement Continuous Integration. Each time that the repository is changed, CI-server has to automatically trigger a build in order to perform all the repetitive tasks that are configured through the build script such as running automated tests etc. A very important feature of the CI-server is the feedback mechanism which notifies the development team in case an error has occurred. This can be done by generating emails or by having a UI that shows signs for each state of the build. A merge into master is completed only when the build is successful and till this happen, the developer who did the merge is responsible to fix any issues that occurred.

Build script

The build script is responsible for the following tasks:

- Automatically checks that specific project related rules are fulfilled.
- Fetch the code from the repository that is set to monitor and compile the source code.
- Ability to integrate with a database.
- Run the suite of tests or configure it in order to use test prioritization and run specific tests in each build script. A general rule of thumb is to create 10 minutes builds.

- Deployment of the software in a UAT server or even in production (e.g. Continuous Deployment practice)

5.6 What are the benefits of using CI?

The most important benefit of Continuous Integration is reduced risk [21], [24]. Continuous Integration solves this problem by integrating the different parts of the project several times in a short timeframe. It provides the development team with a quick answer about the current status of the code. Ensures that the software is working as expected which is something that improves the quality of the product. In order to gain this benefit though should be ensured that the test suite contains well written tests which have value. CI provides a history and statistical data in order to use this data to improve development processes such as version control strategy or the testing suite. Human mistake is also an issue that CI tackles by automating every process that a human would be involved otherwise. Psychological factor is also something that can affect the motivation of the development team which is also known as Broken Windows syndrome.

Continuous Integration decreases the repetitive manual processes which causes saving time and effort. On the other hand, automates code compilation, testing, deployment and feedback. The development team is able to be focused on tasks that provide value to the product and the customers.
Another valuable feature that Continuous Integration provides is the ability to generate deployable software and use it in order to implement Continuous Delivery or Deployment later on. Frequent releases help to eliminate waiting time caused by slow cycles and new features can be shipped to the customers frequently. The absence of CI can cause issues due to the fact that testing happens at a late time and there is the possibility to find many bugs.

Continuous Integration offers project transparency and efficiency to the team during development. Another main CI benefit is that it establishes greater product confidence and improves the trust between the development team and the stakeholders but also between the users and the company.

Developers strongly believe that projects which implement CI are able to give the maximum value to automated tests due to the fact that they run frequently. The automated tests have high quality and they run on every new change that happens in the code so that the development team can be aware when something goes wrong and has to be fixed. Developers believe that projects with CI have higher code quality also except from tests. CI allows developers to focus more on being productive, and to let the CI monitor repetitive steps. Many benefits of CI are also benefits that are inherited from automated testing, test-suite maintenance, code quality etc.

5.7 What are the tradeoffs of using CI?

Despite CI being so popular practice in software engineering, there are many tradeoffs that should be addressed in order to avoid common errors but also it is important to be known to researchers who are trying to find ways to improve this practice. Based on our research, we identify three trade-offs that are connected with Continuous Integration practice [21].

The first tradeoff is about the quality of tests that are written by developers or by software engineers in test (SET). Value should be the main factor to consider when writing tests due to the fact that performance factor is closely connected with that. For every test that the test suite contains and runs during a Continuous Integration build, there are some resources that are used. Low quality tests can cause long builds that cannot give quick feedback. The test suite should be maintained and suspicious tests should be improved because improving the test suite by increasing code coverage or increasing test quality provides huge benefits [24].

Another question is how many tests should be written and how long a build should last. Test prioritization and selection methods should be followed in order to tackle these issues [26, 27]. Researchers should investigate the trade-offs that are related to speed improvements but also percentages of certainty in order to find the best balance between these. Sometimes build duration matters more than other factors and it depends on the type of software that is being developed. Our findings show that developers find it important to keep build times under 10 minutes, which is a general rule of thumb that is proposed by Martin Fowler.

Developers should have in mind also security concerns that extra access to the CI pipeline introduces. CI systems should provide developers with the ability to have more access to the build pipeline, without compromising the security of the system. Access controls and permission strategy should be decided in order to avoid major issues that are produced by security hauls. Researchers should investigate the security challenges that are connected with CI. Although CI aims at automating and simplifying the testing and validation process, the increased infrastructure provides additional attack vectors that can be exploited. They should also propose ways in order to create systems that allow developers to safely expose their CI systems without compromising security.

Developers should always have in mind when they design CI systems or processes that they should avoid custom development processes because of the fact that they increase complexity, maintenance costs etc. Developers should follow agile methodologies and best practices for their processes to be simple and they should consider the long-term costs of highly complex custom CI systems. If the CI becomes overly complex, then it is impossible for an entire team to have knowledge on maintaining it so there should be a person who will be the administrator. This is an unwanted situation because knowledge should be shared among the team in order to be multifunctional and avoid issues when a person leaves the team or the project.

CI tools should be designed with a way that allow developers to use the user interface in order to configure the system, but also give the opportunity to them to choose the type of output that those configurations will provide (e.g. in simple text files that can be included in a version control system). Researchers should aim to collect empirical evidence that will be useful to developers, who wish to reduce complexity by prioritizing convention over configuration. Also, researchers should evaluate the claims of developers who strongly believe that CI improves test quality and increases productivity.

5.8 What are the barriers of using CI?

How easy is to implement Continuous Integration? In reality there are many barriers that make the implementation of CI challenging. We present the most common below [21], [24].

Troubleshooting failing builds

Many times during the development phase there is a need to troubleshoot a build due to failure. Developers can quickly identify the root cause and fix the issue but sometimes this is not the case. Different environments can have different behaviors e.g. maybe the tests pass locally but not in a UAT server. Access rights are a reason that makes troubleshooting a challenging task. CI-server produces reports after build completion which is a really useful feature for developers during troubleshooting. Better logging and storing test artifacts is a common feature of CI servers and so it is a way to make it easier to examine failures.

Long build times

As we mentioned earlier, build times is a very important factor of the CI pipeline. When a pipeline is building no one can proceed with merging his/her changes into the code base so this is a blocking step for developers. The main reason for that is the fact that it is a bad practice to accept changes before ensuring that the status of the previous state is healthy. When this blocking step becomes too long, it reduces developers' productivity and this is a reason that Martin Fowler proposes 10 minute builds. Long build times are a common barrier faced by developers using CI.

Automating the build

Migrating the manual processes to automated builds is a process that requires much effort, time and technical knowledge. DevOps skills are required in order to establish a well configured CI system which will provide the benefits that CI offers.

Setting up and maintaining a CI server or service

A dedicated DevOps team is required in order to setup and maintain CI server and ensure that the system is available to the development team. Managing CI pipelines and resolving any issues

requires dedicated people with the appropriate technical skills which is not always visible especially on low budget projects.

Security and access controls

The security policy that the company has established should be also consider CI pipelines due to the fact that anyone who has access on these he/she also has access to the entire source code. As we mentioned earlier when we discussed the tradeoffs, developers encounter increased complexity, effort that is required and new security concerns when working with CI.

5.9 What are the developers' needs and motivation of using CI?

Regarding needs and motivation there is a contradiction between developers' preferences that have to do with ability of configuration and simplicity. Making a system highly configurable increases complexity though [21, 24]. Below are some of developers' needs.

Easier configuration of CI servers or services

CI tools provide user interfaces and features that make configuration easy most of the times. Software companies hire DevOps engineers to ensure that existed CI systems are configured correctly, and to help set up new CI systems. Open-source developers often use CI as a service, which allows for a much simpler configuration. The absence of people who have the right skills can make the configuration of a CI server difficult.

Better container/virtualization support

Containers are packages that help on virtual isolation without needing virtual machines in order to deploy and run applications. Clean environments are a prerequisite for CI builds so that it is ensured that there aren't any side effects with previous builds that are finished. Docker is a

computer program that supports virtualization/containerization and it is a very good solution to tackle such issues nowadays.

UI importance

Logs are one way to help on debugging issues and fix them. Sometimes this is not an easy job though because logs can be very large and difficult to handle. User interfaces is a really helpful tool for developers so that they can use it along with logs to make their job easier. Another challenge is the way that developers can administer CI tools. Scripts are a common technique for solving that but user interface is also vital in order to provide the developers with the ability to edit these configurations easier.

- **What is the motivation to use CI ?**

CU helps on bug detection

Bugs detection is far easier when using CI. Running the tests frequently helps to tackle this stressful and challenging task. During development phase developers have enough time to fix any detected issues and this fact is more preferable comparing with having to fix bugs that are detected in the production environment [24].

CI helps on build breakages

Kerzazi reported that for one project, up to 2,300 man-hours were lost over a six month period due to broken builds [31]. When a build breaks, the developer who merged the latest changes that caused the build failure is responsible to resolve the issue or rollback his changes. This is not an easy task though since the developer is not always available to work on that (e.g. annual leave etc). These issues can be really blocking for the whole development team. CI helps with several ways to tackle these issues and worry less about such breakages.

CI helps projects deploy more often and have smaller/faster iterations

Developers believe that CI offers them the ability to have shorter development cycles than they otherwise would have and also the benefits of running the suite often plays a vital role to achieve that. Developers can quickly identify breakages and fix them. Feedback enables much faster development cycles and as a result fewer bugs are produced.

CI makes integration easier by enforcing specific workflows

Prior to CI, there was not a way for tools to enforce a specific workflow (e.g., ensuring all tests are run before accepting changes). Version control systems are the favorite solution to enforce a common workflow among the developers of a team. For many developers VCS is the solution to difficult integrations, not the CI. Communication is not an easy task when it comes to technical actions that several members have to do. CI tackles this challenge not only by running all the tests run when a new change is merged into master branch, but letting everyone know what the results are and if the merge was successful and didn't brake anything. Everyone on the team is aware when a code breaks the tests or the build, without having to fetch the code and run it locally. Team awareness is increased, communication is improved and quality is ensured.

6. Common Challenges of adopting CI

The adoption of Continuous Integration practices has a wide range of challenges [25, 28]. In order to obtain the benefits of this practice, the teams need to put extra effort due to the fact that collaboration and coordination challenges are appearing between teams. Team awareness and transparency are sometimes characteristics that need time to be established.

Another challenge when trying to adopt Continuous Integration is the cost. Lack of investment is a challenge in any organization. Sometimes the company may need to hire DevOps engineers in order to set up Continuous Integration server. Lack of the required skills or experience can cause a significant gap.

In addition to these, building complex applications that aim to be frequently released to customers may cause some team members to feel more stress and to require extra efforts from them. A transition to Continuous Integration and frequent releases is a challenge and need time as an approach to be adopted by the development team.

Sometimes the limitations of existing tools and technologies can prevent the developers from achieving the goals of continuous practices. The test suite should be carefully checked in order to see if there is a need to do major changes but also the other parts of the CI should be also analyzed. The absence of specific workflows can produce also several challenges. Resolving conflicts during code integration is a result of this absence and also it is a critical process that many times produces unwanted results and even breakages. From a technical perspective conflicts are caused due to highly coupled design and low quality implementations.

Finally, the organizational culture is difficult to change and every change needs time to be adopted by the people and be a habit. Lack of agile ambassadors and a suitable business model in the company could result in negative consequences for Continuous Integration.

There are several challenges that are closely connected with testing and we will try to analyze them. First of all the lack of proper testing strategy is something that should be tackled as soon as

possible due to the fact that it blocks the adoption of continuous integration. Lack of automated testing and test driven development are also big challenges. Poor test quality is another challenge. Unreliable tests, low code coverage, long running tests and test cases that don't give the desired value are the main reasons that reduce the confidence and block continuous integration/delivery/deployment.

A very interesting and challenging topic in the area of testing is flaky tests. There is a type of tests that often called flaky tests which have non-deterministic behavior. Such behavior is tricky for the development team because they cannot ensure that the tests pass and the functionality works as expected. In situations that all the tests pass, then there is no need to do any further actions to ensure quality and health status of the code. In case of failures though, developers have to trigger again a build in order for the tests to run. The key point behind this process is that a test failure indicates that the merged code introduced an unexpected result in the code. But what if tests fail and during the second build they test pass? Flaky tests can cause several problems during regression testing and in general it is a very important and challenging topic for developers and software engineers in test [29].

One reason that these tests are difficult to tackle is the fact that this kind of tests are non-deterministic. They need hours and sometimes days in order to find the root cause of the problem. Developers need to debug after triggering many builds in order to find a hint which will help them to resolve the issue. Flaky tests can hide bugs in the code and many times there is a tendency to ignore this behavior.

The most common approach is to run a flaky test multiple times or a subset of tests that are connected together in order to have a more clear view of what is happening. In Google, a failing test is re-run 10 times against the same code, and if it passes in any of those 10 re-runs, it is labeled as a flaky test. Another approach is to remove flaky tests from the test suite, but sometimes having a flaky test is better than having no test at all. We should mention that those approaches are more like workarounds than complete solutions that solve the flakiness.

What are the common causes of flakiness?

Sometimes the test execution makes an asynchronous call and does not properly wait for the result of the call to become available before using it. Asynchronous calls do not have a specific number of seconds that are completed so this behavior can cause to have non-deterministic results. Another possible issue is when there are different threads interacting in a non-desirable manner or when the test order matters. More specifically sometimes a test result depends on the order that the tests are run. In this situation the root cause is the fact that there is no proper isolation and the tests are not independent to each other. There are many cases that having a clean environment every time that those tests are run solves such issues.

In addition to these, memory leaks or database issues can cause resource leaks due to the application not managing correctly the resources. Of course the network is also a crucial area which may cause flakiness. Network uncertainties can influence the behavior of tests and for this reason the developers should consider this case when writing tests. Another common cause is when relying on the system time to run the tests. Some tests may fail because of different time zones between the actual server and the local server when these two exist in different countries.

How to tackle flaky test failures?

Tackling the flakiness of tests is the first step in order to fix them. The first step to identify if a test is flaky is to run the test multiple times. If there is a non-deterministic result then it is clear that the test is flaky. In this case though, we have to mention that rerunning the tests multiple times is time consuming, and the problem is more serious especially if there are multiple test failures.

Another way to have a more clear understanding about flaky tests is to classify the flakiness. For example a test may pass in a specific platform and fail in another one. This category is not about issues that we previously discussed and had to do with environment but it is about for example the operating system that the test suite is run or the version of Java etc. For this reason

developers should strive to write tests that are platform independent. An example of platform dependence is a test failing due to a different order of files on a specific file system.

There are several times that there is a need to add a “sleep” or “waitFor” method in order to force a time delay in a test. A sleep is a way to pause the current thread for a specific timeframe that we would like. The other method that we mentioned, “waitFor”, is a set of methods used to wait for some condition to become true. There are many flaky tests that can be tackled by adding more time delay. Although it is a method that solves issues, we would like to mention that the performance of the test suite is influenced by such add-ons. The duration of the build is the next important topic that can be influenced by such changes.

To conclude, there are many common fixing strategies that can help the developers or the software engineers in test to fix flaky tests. Except from those that we mentioned above, reordering code and making the code deterministic can help to solve such issues. Changing assertions in order to consider all valid values can also be a solution. Finally merging tests is a good practice in order to keep or increase the code coverage and tackle flakiness. Regression testing is important but can lose much of its value due to flaky tests. While there is no one solution for every problem and which will be able to address all categories of flaky tests, there are broad enough categories for which it should be feasible to develop automated solutions to tackle and fix flaky tests.

7. Practices of Continuous Integration

Due to high competition companies pay significant attention and allocate resources to develop and deliver high-quality software in less time. Continuous Integration (CI), Continuous Delivery (CDE), and Continuous Deployment (CD), are some key practices that aim to help organizations to improve every phase of their development cycle and as a result to be able to deliver on time and according to the plan software features without compromising quality.

The most popular are the following [22, 24, 28]:

Maintain a single source repository

A single source repository is the foundation in every software product. Unfortunately there are still many software products that are developed without using such a popular and useful practice. A software project includes many files and many people who modify them during the development phase. A version control system is one of the most useful tools that can host repositories in order to store all the parts that are needed in order to run a project. Test scripts, property files, database schema are just some of them. Everyone should build the project only after cloning the repository and do a checkout on the required branch. A proper branching strategy should be established before the actual implementation of features starts. The most important thing is to maintain only a single source repository in order to have a single point of truth.

Automate the build

Automation is one of the crucial parts of a CI project. All the repetitive tasks should be automated and of course an automated test suite should be included in these. All the members of the development team should be able to clone the repository and checkout the master branch and after running the required build commands to be able to run a build. Build automation is a

practice most organizations have adopted. However, many teams decide to use CI in a way that it's more relevant to scheduled builds (i.e. every night), or continuous builds but they are not actually CI builds. Without validation of every build, there is not continuous integration.

Making the build self-testing

A build contains several repetitive steps such as compiling, linking, and everything else that is required to run an application. The build may be successful but are we sure that the application works as expected? A good way to ensure it is to include the execution of automated tests in order to be able to detect bugs more quickly. Self testing code is a characteristic that test driven development has adopted and this is a reason that developers are trying to apply such a practice and gain its benefits. In order to run the automated tests during each build, there is a need to have an automated testing suite that can be executed with a simple command. A successful build is the one that no tests failed and a self-testing build is able to gain such a benefit. Tests don't prove the absence of bugs but writing tests and run them frequently even if they are not covering all the cases tests, is much better than having a code base that does not include tests.

Keeping the branch in sync with the mainline

Every developer should keep his/her branch in sync with the master branch. Resolving conflicts is a painful process which may result into undesired situations like introduction of new bugs or broken code. By syncing the local branch with the mainline frequently, developers can quickly identify that there are conflicts between their code and another part of code. Fixing such problems is far easier than having conflicts that are undetected for weeks.

As a best practice it is said that every developer should commit his/her changes to the repository every day. Frequent commits encourage developers to break down their work into small parts that contain a complete part of a specific functionality.

Every commit into the mainline should trigger a build in an integration environment

Commits into the mainline are one of the most critical steps during software development. Some common reasons that can cause code breakages are the following: Discipline, has to do when developers do not update their local branch from the mainline and also they don't build locally before they commit. Another reason is differences between developers' local machines. This fact may cause functionality to work as expected in one developer's machine and not working to another. We should ensure that regular builds happen on an integration machine and only if this integration build succeeds should the commit be considered to be done. The developer who commits is responsible for monitoring the progress of the build and fix any issues that were caused by his/her changes.

The manual build approach is really simple since the developer checks out the mainline and triggers a build. Automated builds are a little more complex. A continuous integration server monitors the repository and checks for possible changes. When a new commit against the mainline is detected, the server automatically triggers a build, and notifies, usually via email, the developer who did the commit.

Many companies do regular builds based on a schedule. This is considered as a different approach that is not the same thing as a continuous build and of course it is not enough for continuous integration. The goal of continuous integration is to detect issues as soon as possible and for this reason, when a build fails it should be fixed right away in order to continue working on a stable base.

Keep the build fast

The huge benefit of Continuous Integration is the fact that it provides quick feedback. In order to use this feedback there should be build that does not last too long until it is completed. The main goal is to try keeping a build to last no more than 10 minutes. By achieving such fast builds the developers can receive the feedback and either quickly fix any issues or proceed with committing

new changes in the mainline. If the build lasts too long then this fact blocks the process and the development team has to be idle until the end of the build.

Test in a production-like environment

The main reason of testing is to quickly detect any issues that the application will have in production. An important prerequisite is to have available a production-like environment on which the production version of the application will run. If we test the application using a different environment, every difference is a risk that may cause different behaviors across the environment that we use and the environment that the application will finally be deployed (production).

A duplicate production environment is the best way to ensure that there will be no differences. It is not always an easy task though to have such an environment available especially when the application and the infrastructure are complex. For this reason it is common to have a quality assurance environment which is close to the production environment in order to perform most of the testing without having big delays, and use a production clone for secondary testing that will be done occasionally (e.g. during performance testing or release testing).

Monitoring and providing feedback

Continuous Integration is also a way of communication in a development team. Everyone should easily see the health status of the application. One of the most important things is to have a clear view on the state of the mainline build. CI servers can visualize such information in order to give feedback to the development team. Usually there are dashboards that present information about who triggered the current build, the time that has past and also the time that is remaining till the completion of the build. Also CI servers provide a history of changes but the most useful characteristics are logging and status indicators so that developers can debug but also know the current health status of every build. Remote teams can also benefit from all the above advantages.

8. Continuous Integration at Google

Many huge organizations like Google, give much attention on having always the ability to quickly indentify and fix any issues when the code changes during software development. Since the code base is huge, they have large scale repositories which they maintain regularly by having small and rapid development cycles. The process of identifying and fixing such issues is one of the most time consuming tasks in the software development life-cycle. For that reason, there is a high demand for implementing automated techniques in order to help the development team to identify such changes while minimizing manual human intervention. There are various techniques that have recently been proposed to identify such code changes. However, these techniques are more suitable on smaller projects due to having short comings and this fact make them unsuitable for rapid development cycles like Google has.

In order to better understand how huge is the software that Google develops and maintains, we would like to present some details on that. According to [33], “The Google codebase includes approximately one billion files and has a history of approximately 35 million commits spanning Google’s entire 18-year existence. The repository contains 86TB of data, including approximately two billion lines of code in nine million unique source files. The total number of files also includes source files copied into release branches, files that are deleted at the latest revision, configuration files, documentation, and supporting data files. In 2014, approximately 15 million lines of code were changed in approximately 250,000 files in the Google repository on a weekly basis. Google’s codebase is shared by more than 25,000 Google software developers from dozens of offices in countries around the world. On a typical workday, they commit 16,000 changes to the code base and another 24,000 changes are committed by automated systems. Each day the repository serves billions of file read requests, with approximately 800,000 queries per second during peak traffic and an average of approximately 500,000 queries per second each workday. Most of this traffic originates from Google’s distributed build and-test systems.”

Reliability is a crucial factor during development of such kind of software, Google has adopted Continuous Integration (CI), where each time that a piece of code changes triggers a job in order for code to be compiled and tests to be executed so that regressions can be caught earlier in the development lifecycle.

Figure 11 shows the code repository and development workflow at Google. The Google code repository does not use multiple branches, i.e. there is a single branch for the entire code base, called HEAD, where all developers submit their changes.

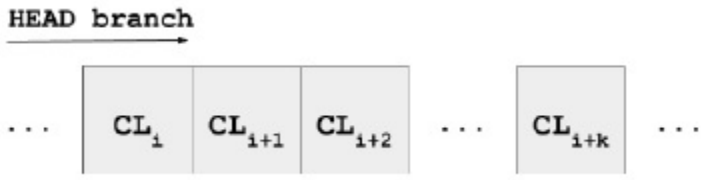


Figure 11. Code repository and development workflow at Google [33].

In order to provide details about this process, we would like to start from the fact that in Google, changes are submitted to the code repository in atomic units called change-lists (CLs). Each change-list is submitted to the same code repository branch that is called HEAD. When the developers change the code in the repository, they submit such an atomic change to the code base, called a change-list (CL). So every CL submission results in a new system version and it is validated by executing test. There are different types of tests and for that reason some tests are run before the CL is submitted, while some tests are run after the submission. When a test fails after a CL is submitted then it is said that there is a regression issue. When a CL introduces such a regression into HEAD that result in test failures and it is important to identify it quickly, so that the development team can fix it and the velocity is not disrupted.

In summary, for code repositories where there is rapid evolution as at Google, finding the system version where a regression was first introduced is the crucial first step in debugging failing tests. Tests are classified into four categories at Google according to their size: small, medium, large, enormous (Figure 12). Size refers to the execution time of the test. According to [33], at Google, given the size of each test, there are two typical development workflows regarding CLs where tests are executed:

1) Pre-submit tests: Tests are run before a CL is submitted. If all tests are successful, then the CL is allowed to be submitted. Pre-submit tests are small/medium tests (although large tests are also allowed), so that developers are not blocked for a long time to get the results of test execution before submitting a CL.

2) Post-submit tests: In this workflow, a CL is submitted to the repository without waiting on the results of test execution. Tests are executed periodically to check if any of the submitted CLs caused a regression, and if so, developers are notified of the regression after the submission in order to be able to fix that quickly. Typical post-submit tests are large/enormous tests, because such tests would take too long to execute and hinder development velocity if they were to be executed before CL submission.

	Runtime Limit	Example Tests
Small	1 min	Unit
Medium	5 min	Unit / Integration
Large	15 min	System / Integration / End-to-end
Enormous	No limit	System / Integration / End-to-end

Figure 12. Classification of tests at Google according to their size [33].

Google workflow

Google's repository is based on monolithic architecture and it provides a single point of truth for thousands of developers around the world. Google uses a custom, homegrown version-control system in order to be able to host its large codebase which is visible and can be used from most of the software developers in the company.

Several best practices and supporting systems are required to avoid constant breakage in an environment where thousands of engineers commit thousands of changes to the repository on a daily basis. Google has an automated testing infrastructure that initiates a rebuild of all affected dependencies on almost every change committed to the repository. If a change introduces a wide build breakage, the system is configured so that it can automatically undo the change. To reduce of such situations where bad code is being committed, the highly customizable Google "pre-

submit" infrastructure provides automated testing and analysis of changes before they are added to the codebase. One of the most important aspects of Google is its culture that encourages code quality methods and best practices as there is the expectation that all code is reviewed before being committed to the repository. Most developers can view and propose changes to files anywhere across the entire codebase, with the exception of a small set of highly confidential code that is more carefully controlled. A change in code often receives a detailed code review from one developer, evaluating the quality of the change, and a commit approval from an owner, evaluating the appropriateness of the change to their area of the code base. Code reviewers comment on aspects of code quality, including design, functionality, complexity, testing, naming, comment quality and code style as documented by the various language-specific Google style guides. In summary, Google has developed a number of practices and tools to support its enormous monolithic codebase and it is given much attention on reviewing any change that aims to be committed in the repository.

9. Continuous Integration Tools

Continuous Integration has been around for a while now, so in order to support this practice, there is a need for CI tools [28, 29, 32]. In the context of this study we have focused primarily on open source continuous integration tools but also some commercial. Open source CI tools are often available for free, which allows us to do some experiments in order to see which tools is better fit our needs without requiring a financial investment. Additionally, most of them are open source so we have the access to the entire code base, in order to be able to customize the tool if we need.

Continuous Integration tools help a development team to stick to the quality standards that it has already defines, by running tests every time a new commit is pushed into the repository and also report the results. Combined with continuous delivery tools, code can be tested on multiple configurations, run additional performance tests, and automate every step until the final deployment to a production server.

The tools that may be useful depend on various factors including:

- Programming language and application architecture
- Operating system and browsers support strategy
- Experience and skills of the team
- Scaling plans
- Delivery goals

Version Control Systems

The most popular version control systems used in continuous integration pipelines are Subversion (SVN) and GIT. Subversion is very easy to understand and it does not have a big learning curve in order to use it. GIT is the most popular version control system and it is closely connected with GitHub. Both systems are very similar with the basic difference to be the way

they handle repositories. SVN has a single repository whereas GIT uses multiple repositories based on the work that is done by each developer.

CI-servers

Jenkins

Jenkins is the most popular open-source Continuous Integration server based on Java platform. The fact that it is an open-source tool and it has the ability for extensions is the main factor of success. Kohsuke Kawaguchi is its founder (2004) and, at first it was called Hudson. Jenkins is used by well-known companies such as Facebook, Yahoo, Ebay, Netflix.

According to the official website (https://jenkins.io/), Jenkins offers the following features:

- Easy installation and configuration via a web rich user interface.
- Plug-ins: Jenkins has a large plug-in ecosystem with hundreds of them which offers easy integration with any tool in the CI and CD tool chain.
- Extensibility: Jenkins can be extended and modified, and it is easy to create new Jenkins plug-ins via its plug-in architecture in order to fit project needs.
- Distributed builds: Jenkins can distribute build/test loads to multiple machines running under different operating systems (OS X, Linux, and Windows).

One advantage of Jenkins is the fact that it has a user interface dashboard that can be used to do several actions such as project modification, scheduling and reporting. Jenkins also provides customizable user notifications via multiple ways such as via email.

Bamboo

Bamboo is a software which helps to apply Continuous Integration practices and it is developed by Atlassian (https://www.atlassian.com/software/bamboo). Some features that Bamboo has are the following:

- Easy build import from popular open source tools and native support for GIT and SVN.

- Easy set up CI builds. Supports scripts, smoke tests, and third-party technologies to define the deployment steps for each environment, from continuous integration, to deployment, to delivery.
- Easy migration from Jenkins.
- Buil-in GIT branching and workflows. In addition to GIT, Bamboo hooks into SVN, Mercurial, Perforce, CVS, and repositories in Bitbucket and Fisheye. Supports commit messages, authors, reference numbers, and dates but also it is visible to see diffs, history, and browse related code.
- Enterprise support and resources. Provides a variety of training, best practices, and support resources but also a support team that can solve issues on demand. The following companies use Bamboo: Cisco, BMW, NASA.

Team City

Team City is a multifunctional Continuous Integration server, which can be described by easy installation. It supports many version control systems, authentication, deployment and testing without having to configure anything. It supports project hierarchy in order to be able to form the project tree to inherit settings from the parent. Another feature is templates that can create common settings and inherit various configurations from it. Team City is extensible and is easy to setup but also free for small teams and open source projects. A very useful feature is the fact that it supports on-the-fly build progress reporting so that the team doesn't have to wait for a build to finish in order to discover that something went wrong. Team City is used by following known companies: Apple, Stack Overflow.

Travis CI

Travis CI is a powerful hosted CI service that is free to open source projects and is fully integrated with GitHub. It supports a wide range of languages (over 30) and provides a quick setup. Travis CI supports pull request and also live build views. In addition, it provides auto deployments on passing builds and clean virtual machines for every build. It supports various operating systems such as Mac, Linux and iOS.

Choice Criteria for Optimal Continuous Integration Tool

The selection of the right tools is a very hard task that a company faces because of the fast evolve and the big variety of choices that exist. Many times the requirements and needs aren't clear in a project and for that reason it is tricky to support the process of Continuous Integration. It is important to find out what characteristics a CI tool must have [28, 29, 32].

The most important factors during this process are to consider time and cost. Time can influence cost so a CI tool is considered appropriate when it can serve the team of developers in many projects, rather than in one. Of course the development team might want to change a tool in the middle of the development process but in any case this is not a desired situation because it is not always an easy job.

Functionality is also a significant factor when choosing a CI tool. By saying functionality we mean the following:

- Build execution

  A CI system must be polling-driven, whereas schedule-driven with pre-determined schedule for build execution is not a true Continuous Integration tool. Event-driven execution is also a possible option which means that the build is performed after any changes are made in the repository.
- Maintaining the version control system

  The team should pay attention on checking how a CI tool interacts with methods that the repository uses. CI server has to recognize changed files and their versions and integrate in a very good way with the VCS.
- Feedback

  Different types of feedback are supported by CI (User interface, email notifications etc).
- User interface

  A well designed interface can save time while working with CI tool. User interface can be provided in different forms; for instance, it can be a web application interface.

- Built-in security features

  Provide authentication and authorization, which allow controlling the access to results review and to modify the configuration.
- Extensibility

  The possibility to extend functionality can be a significant factor in choosing CI tool.
- Build tool integration
- Build labeling

Compatibility is a characteristic that defines how well a CI tool can integrate with other elements of the development process. Firstly, the CI tools should be aligned with the build configuration. Secondly, installation of additional software or plug-ins may be challenging. Thirdly, programming language compatibility is important. Some of the CI tools are created for working with specific programming languages while others can work with any programming language.

Reliability is another important factor. A CI tool that has been tested in various projects and from various teams results on having a good reputation which makes it more reliable than a tool which is still in beta-version state and just appeared in the market. Another crucial factor is communities related to this tool. If the size of community is big with large amount of developers and users, it can indicate that there are a lot of answers to questions and possibly a very good documentation. This fact means that it would be easy to find solutions to problems.

Longevity is a factor that concerns the future of the tool. As it stands for reliability also, the size of community and popularity between users and developers is important. We can see that longevity is higher regarding tools with open source code, since the community of users is constantly updating them. Furthermore, a good CI tool with diverse functionality is used longer than a tool with poor functionality.

Another important criterion is usability. A well designed user interface is a crucial factor when it comes to usability. The learning curve that it needs should be minor and everyone should be able to be familiar with the tool in short time. Also it should be easy for users to achieve their objective through using the tool.

10. Conclusion

At the time of writing, this is the only research effort focusing on the systematic mapping of Continuous Integration in software engineering. The research presented in this thesis provided a breadth-first review of the research relating to Continuous Integration in software engineering to promote a better understanding of a new and pervasive area. In particular, several fundamental research questions that are relevant to current research efforts focusing on Continuous Integration in software engineering were answered, while also providing an excellent platform for further research and investigation in the area.

In particular, it is logical that future work should focus on the development of systematic and literature reviews that are aligned with the areas of software engineering identified in this study, such as the creation of a systematic review of Continuous Integration in software engineering that is focused on software testing. The combination of these reviews, coupled with the systematic mapping presented in this research, can serve to provide a complete perspective of the primary research relating to Continuous Integration in software engineering.

The goal of this thesis is to investigate Continuous Integration phenomenon, its origins and to understand the needs, benefits, tradeoffs and motivation of using CI. Another goal was to find the latest trends, challenges and practices of CI along with listing the most popular CI tools. We expect that after reading this thesis, results of survey will help the reader to have a solid overview about CI, understand its connection with Agile Methodologies and also get to know the latest CI trends and principles. In order to perform this survey, we feel that we used the most latest and actual Continuous Integration available resources and results of researches.

References

[1] ShubhinderKaur, E. A. (2015). *Process of Moving from Traditional to Agile software Development: A Review.*

[2] Yu Beng Leau, W. K. (2012). *Software Development Life Cycle Agile vs Traditional Approaches.*

[3] Helena H. Olsson, H. A. (2012). *Climbing the "Stairway to Heaven" - A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software.*

[4] Dooley, J. F. (2017). *Software Development, Design and Coding.*

[5] Royce, W. (1987). *Managing the development of large software systems: concepts and techniques.*

[6] S.Balaji, D. M. (2012). *WaterfallLVs V-Model Vs Agile:A comparative study.*

[7]12th state of Agile. (2018). Retrieved 5 28, 2018, from http://stateofagile.versionone.com/

[8] Pedro Serrador, J. K. (2015). *Does Agile work? — A quantitative analysis of agile project success.*

[9] Irum Inayat, S. S. (2015). *A systematic literature review on agile requirements engineering practices and challenges.*

[10] K. Beck, M. B. (2001). *The agile Manifesto.*

[11] Laanti, M. (2014). *Characteristics and Principles of Scaled Agile.*

[12] Schwaber, K. (2004). *Agile Project Management With Scrum.*

[13] Sutherland, J. (1993). *The roots of Scrum: How the japanese experience changed global software development.*

[14] J Sutherland, K. S. *The Scrum guide. the definitive guide to Scrum: The rules of the game.*

[15] Lowell Lindstrom, R. J. *Extreme Programming and Agile Software Development Methodologies.*

[16] C Ebert, G. G. (2016). *DevOps.*

[17] Syed Roohullah Jan, S. T. *An Innovative Approach to Investigate Various Software.*

[18] Itti Hooda, R. S. (2016). *Software Test Process, Testing Types and Techniques.*

[19] Brian Fitzgerald, K.-J. S. (2014). *Continuous Software Engineering and Beyond: Trends & Challenges.*

[20] Pilar Rodríguez, A. H. (2015). *Continuous deployment of software intensive products and services: A systematic mapping study.*

[21] Danny Dig, M. H. (2016). *Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects.*

[22] Fowler, M. (2000). Retrieved 5 28, 2018, from Continuous Integration: https://www.martinfowler.com/articles/originalContinuousIntegration.html

[23] Kai Petersen, R. F. (2008). *Systematic Mapping Studies in Software Engineering.*

[24] Hilton, M. (2017). *Trade-Offs in Continuous Integration: Assurance, Security, and Flexibility.*

[25] Darko Marinov, A. Z. (2014). *Balancing Trade-Offs in Test-Suite Reduction.*

[26] Nima Dini, A. S. (2016). *The Effect of Test Suite Type on Regression Test Selection.*

[27] S.Yoo, M. (2012). *Regression testing minimization, selection and prioritization: a survey.*

[28] Mojtaba Shahin, M. A. (2017). *Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices.*

[29] Qingzhou Luo, F. H. *An Empirical Analysis of Flaky Tests.*

[30] Meyer, M. (2014). *Continuous Integration and Its Tools.*

[31] Noureddine Kerzazi, B. A. (2016). *Botched Releases: Do We Need to Roll Back? Empirical Study on a Commercial Web App.*

[32] Polkhovskiy, D. (2016). *Comparison between Continuous Integration tools.*

[33] Celal Zeftci, J. R. (2017). *Who broke the build?Automatically identifying changes that induce test failures in CI at Google Scale.*

[34] En.wikipedia.org. (2018).Comparison of continuous integration software. [online] Available at: https://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software [Accessed 7 Jun. 2018].

[35] Haneen Anjum, Maham Khan, Zainab Shahid (2017). *A Comparative Analysis of Quality Assurance of Mobile Applications using Automated Testing Tools*

[36] Nilsson, A., Bosch, J. and Berger, C. (2014).Visualizing Testing Activities to Support Continuous Integration: A Multiple Case Study.

[37] Bhattacharya, A. (2014). Impact of Continuous Integration on Software Quality and Productivity.

[38] Fitzgerald, B. and Stol, K. (2014). Continuous software engineering and beyond: trends and challenges.

[39] Philipp Diebold, Marc Dahlem (2014). Agile practices in practice.

[40] Medeiros de Campos, J., Arcuri, A., Fraser, G. and Maranhão de Abreu, R. (2014).Continuous test generation.

[41] Vasilescu, B., Van Schuylenburg, S., Wulms, J., Serebrenik, A. and van den Brand, M. (2014). Continuous Integration in a Social-Coding World: Empirical Evidence from GitHub.

[42] Elbaum, S., Rothermel, G. and Penix, J. (2014).Techniques for improving regression testing in continuous integration development environments.

[43] Ståhl, D. and Bosch, J. (2014).Modeling continuous integration practice differences in industry software development.Journal of Systems and Software.

[44] Debbiche, A., Dienér, M. and BerntssonSvensson, R. (2014). Challenges When Adopting Continuous Integration: A Case Study.

[45] Brandtner, M., Giger, E. and Gall, H. (2014).Supporting continuous integration by mashing-up software quality information.

[46] Ståhl, D. and Bosch, J. (2014).Continuous Integration Flows.Continuous Software Engineering.

[47] Geiger, C., Przytarski, D. and Thullner, S. (2014). Performance testing in continuous integration environments.

[49] Soni, M. (2015). End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery.

[50] Laukkanen, E., Paasivaara, M. and Arvonen, T. (2015).Stakeholder Perceptions of the Adoption of Continuous Integration -- A Case Study.

[51] Seth, N. and Khare, R. (2015). ACI (automated Continuous Integration) using Jenkins: Key for successful embedded Software development.

[52] Anderson, C. (2015). Docker [Software engineering].

[53] Waller, J., Ehmke, N. and Hasselbring, W. (2015).Including Performance Benchmarks into Continuous Integration to Enable DevOps.

[54] Vasilescu, B., Yu, Y., Wang, H., Devanbu, P. and Filkov, V. (2015).Quality and productivity outcomes relating to continuous integration in GitHub.

[55] Virmani, M. (2015). Understanding DevOps & bridging the gap from continuous integration to continuous delivery.

[56] Knauss, E., Staron, M., Meding, W., Soder, O., Nilsson, A. and Castell, M. (2015).Supporting Continuous Integration by Code-Churn Based Test Selection.

[57] Rathod, N. and Surve, A. (2015). Test orchestration a framework for Continuous Integration and Continuous deployment.

[58] Lai, S. and Leu, F. (2015).Applying Continuous Integration for Reducing Web Applications Development Risks.

[59] Gambi, A., Rostyslav, Z. and Dustdar, S. (2015). Improving Cloud-Based Continuous Integration Environments.

[60] Claps, G., BerntssonSvensson, R. and Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way.

[61] Laukkanen, E. and Mantyla, M. (2015). Build Waiting Time in Continuous Integration -- An Initial Interdisciplinary Literature Review.

[62] Boettiger, C. (2015). An introduction to Docker for reproducible research.

[63] Shrivastava, S. and Rathod, U. (2015). Categorization of risk factors for distributed agile projects.

[64] Lwakatare, L., Kuvaja, P. and Oivo, M. (2015).Dimensions of DevOps.

[65] Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M. and Steinder, M. (2015).Performance Evaluation of Microservices Architectures Using Containers.

[66] J.Engblom (2015) Virtual to the (near) end: Using virtual platforms for continuous integration.

[67] Chin-Yun Hsieh, Chien-Tsun Chen (2015). Patterns for CI Builds in Cross-Platform Agile Software Development.

[68] Kivanc Muslu, Yuri Brun, Alexandra Meliou (2015). Preventing data errors with continuous testing.

[69] Johannes Gmeiner, Rudolf Ramler (2015). Automated testing in the continuous delivery pipeline: A case study of an online company.

[70] Bosch, J. (2016). Speed, Data, and Ecosystems: The Future of Software Engineering. IEEE Software.

[71] Adams, B. and McIntosh, S. (2016). Modern Release Engineering in a Nutshell -- Why Researchers Should Care.

[72] Dingsøyr, T. and Lassenius, C. (2016). Emerging themes in agile software development: Introduction to the special section on continuous value delivery.

[73] Bolduc, C. (2016). Lessons Learned: Using a Static Analysis Tool within a Continuous Integration System.

[74] Balalaie, A., Heydarnoori, A. and Jamshidi, P. (2016).Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture.

[75] F. Tripp, J. and Armstrong, D. (2016). Agile Methodologies: Organizational Adoption Motives, Tailoring, and Performance.

[76] Hilton, M., Tunnell, T., Huang, K., Marinov, D. and Dig, D. (2016).Usage, costs, and benefits of continuous integration in open-source projects.

[77] Hu, J. och Li, Y. (2016) Implementation and Evaluation of Automatic Prioritization for Continuous Integration Test Cases.

[78] Bani Muhamad, F., Sarno, R., Ahmadiyah, A. and Rochimah, S. (2016). Visual GUI testing in continuous integration environment.

[79] Zúñiga-Prieto, M., Insfran, E., Abrahao, S., & Cano-Genoves, C. (2016).Incremental Integration of Microservices in Cloud Applications.

[80] Schermann, G., Cito, J., Leitner, P. and Gall, H. (2016). Towards quality gates in continuous delivery and deployment.

[81] Balalaie, A., Heydarnoori, A. and Jamshidi, P. (2016). Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture.

[82] Adams, B. and McIntosh, S. (2016). Modern Release Engineering in a Nutshell -- Why Researchers Should Care.

[83] K. Sree Poornalinga, Dr. P. Rajkumar (2016). Continuous Integration, Deployment and Delivery Automation in AWS Cloud Infrastructure.

[84] Kim Dikert, Maria Paasivaara (2016). Challenges and success factors for large scale agile transformations: A systematic literal review.

[84] Ståhl, D., Mårtensson, T. and Bosch, J. (2017). The continuity of continuous integration: Correlations and consequences.

[85] Martensson, T., Stahl, D. and Bosch, J. (2017). Continuous Integration Impediments in Large-Scale Industry Projects.

[86] Lavriv, O., Buhyl, B., Klymash, M. and Grynkevych, G. (2017). Services continuous integration based on modern free infrastructure.

[87] Gautam, A., Vishwasrao, S. and Servant, F. (2017). An Empirical Study of Activity, Popularity, Size, Testing, and Stability in Continuous Integration.

[88] Beller, M., Gousios, G. and Zaidman, A. (2017). Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub.

[89] Shahin, M., Ali Babar, M. and Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices.

[90] Islam, M. and Zibran, M. (2017). Insights into Continuous Integration Build Failures.

[91] Pinto, G., Reboucas, M. and Castor, F. (2017). Inadequate Testing, Time Pressure, and (Over) Confidence: A Tale of Continuous Integration Users.

[92] Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V. and Vasilescu, B. (2017). The impact of continuous integration on other software development practices: A large-scale empirical study.

[93] Laukkanen, E., Itkonen, J. and Lassenius, C. (2017). Problems, causes and solutions when adopting continuous delivery—A systematic literature review.

[94] Mårtensson, T., Ståhl, D. and Bosch, J. (2017). Exploratory Testing of Large-Scale Systems – Testing in the Continuous Integration and Delivery Pipeline.

[95] Hilton, M., Nelson, N., Tunnell, T., Marinov, D. and Dig, D. (2017). Trade-offs in continuous integration: assurance, security, and flexibility.

[96] Ziftci, C. and Reardon, J. (2017). Who broke the build? Automatically identifying changes that induce test failures in continuous integration at Google Scale.

[97] O'Connor, R., Elger, P. and Clarke, P. (2017). Continuous software engineering-A microservices architecture perspective.

[98] Rahman, Akond & Agrawal, Amritanshu & Krishna, Rahul & Sobran, Alexander & Menzies, Tim. (2017). "Doing" Agile versus "Being" Agile. Empirical Results from 250+ Projects.

[99] Fitzgerald, B. and Stol, K. (2017). Continuous software engineering: A roadmap and agenda.

[100] Maudoux, Guillaume , Mens, Kim (2017). Bringing Incremental Builds to Continuous Integration.

[101] Memon, A., Zebao Gao, Bao Nguyen, Dhanda, S., Nickell, E., Siemborski, R. and Micco, J. (2017). Taming Google-scale continuous testing.

[102] Claus Pahl, Jacopo Soldani (2017). Cloud container technologies : A state of the art review.

[103] Rory V.O'Connor, Peter Elger (2017). Continuous software engineering – A microserviecs architercture perspective.

[104] Ioannis K. Moutsatsos,Imtiaz Hossain, Claudia Agarinis (2017). Jenkins CI an open source Continuous Integration system, as a scientific data and image processing platform.

[105] Borle, N., Feghhi, M., Stroulia, E., Greiner, R. and Hindle, A. (2018).Analyzing the effects of test driven development in GitHub.

[106] Gallaba, K. and McIntosh, S. (2018). Use and Misuse of Continuous Integration Features: An Empirical Study of Projects that (mis)use Travis CI.

[107] Liang, J. (2018). Cost-effective techniques for continuous integration testing (master thesis). University of Nebraska-Lincoln, Lincoln, Nebraska, United States.

[108] Liang, J., Elbaum, S. and Rothermel, G. (2018).Redefining prioritization.

[109] Herzig, K. (2018). Testing and continuous integration at scale.

[110] Marijan, D., Liaaen, M. and Sen, S. (2018). DevOps Improvements for Reduced Cycle Times with Integrated Test Optimizations for Continuous Integration.

[111] Ricardo Colomo Palacios, Eduardo Fernandes, Pedro Soto-Acosta, Xabier Larrucea (2018). A case analysis of enabling continuous software deployment through knowledge management.