BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

CONTINUOUS INTEGRATION AND AUTOMATED CODE REVIEW IN OPEN SOURCE PROJECTS

PRŮBĚŽNÁ INTEGRACE A AUTOMATIZOVANÁ KONTROLA KÓDU V PROJEKTECH S OTEVŘENÝM ZDROJOVÝM KÓDEM

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **ADRIÁN TÓTH**
AUTOR PRÁCE

SUPERVISOR **Ing. LENKA TUROŇOVÁ**
VEDOUCÍ PRÁCE

BRNO 2018

Abstract
TODO

Abstrakt
TODO

Keywords
TODO

Klíčová slova
TODO

Reference
TÓTH, Adrián. *Continuous Integration and Automated Code Review in Open Source Projects*. Brno, 2018. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Lenka Turoňová

Continuous Integration and Automated Code Review in Open Source Projects

Declaration

TODO

....

Adrián Tóth

16.05.2018

Acknowledgements

TODO

Contents

Chapter 1

Introduction

In these days, Continuous Integration (CI) is more often used in larger projects, where multiple developers are working on one and the same software product. This process ensures fast software development, called eXtreme Programming (XP), known as agile software development methodology. The methodology is mainly used to accelerate the development, nevertheless, development of software may be disrupted in various other ways. Nowadays, although this type of software development has many disadvantages, it is still much more often used on larger projects. The progress of the development may not be reached with a continuous integration which guarantees less disorder and failures. You may also know that the continuous integration is a part of the following open-source projects e.g. Facebook, Twitter, Mozilla. These projects use one of many famous continuous integration service Travis CI. Excluding Travis CI, there are plenty of other continuous integration services you may heard about, such as Jenkins, TeamCity, CircleCI, GitLab CI, Codeship and so on.

The software development process requires many code checking tools after every single code change in the source code. For as much as with every single change of code, there is a possibility to add, fix, derange or deteriorate any parts of the software product. These tools provide an automated code review and they afford a quick feedback by which they try to prevent these code impairments. Feedback about his adjustment is sent to the developer, who has made the change in the code. The automated process which provides the code review does not bother with executing a huge amount of tests. Above mentioned process is conducted via continuous integration server, which compiles the code, runs scripts and tests. The results are aggregated and the feedback is given to the developer who has made this code change. Continuous integration server is invoked every single time after any change is fetched in the source code and he had to execute the stated acts which are predefined. In next chapters we describe in details how does this workflow work and what steps are required to run.

The essence of this work is about the basics of continuous integration and its fundamentals. This thesis attempts to explain how the fundamentals of continuous integration and automated code review work. It describes how it is integrated to the software development, and how it works on an extensive project nowadays. Examples will be based on open-source project e.g. ManageIQ, which is a cloud manager founded by Red Hat, Inc. The development process of the ManageIQ rests in agility and stability of the progress. These main factors of the development process could not be reached without a quick feedback to the developers working on project about their changes, that are submitted to the software product.

Chapter 2

Continuous Integration

In the face of the fact that continuous integration and automated code review are used in a lot of projects, it is still an unknown part of software development. Despite CI rising as a big success story in automated software engineering, it has received almost no attention from the research community [22]. There are only a few researches describing this part of development how is it deployed, managed and used. Development analysts are not giving an adequate attention to this part of software development. They are usually describing it as a common part of development in a software development process. This part is concerning to extreme programming due to fast code change deployment. This development technique is very adaptive and still more and more open source projects are using it. There are many developers relying on this type of software development which helps them rapidly. This chapter will give you a detailed view of the modern in-use software development methodology which is still in evolution.

2.1 Continuous Integration

Continuous integration (CI) has a key role in the software development process consisting of a few certain unavoidable steps which will be described later in next sections. CI is believed to be an effective way to integrate the source code faster and certify the result of such cooperation, hence an important component of modern software parallel development environments [15]. Everything begins at the moment, when one developer who has made changes in a source code of the software product is trying to commit them into the software product. The process of continuous integration has begun at this point and lasts until feedback is sent back to the developer. These stages of continuous integrations are proceed every time after the CI server has detected a change in a version control repository. This automation has a lot of benefits which are necessary to keep the software product without any kind of defects. Many of them are detected in time and reported back to the developer as a corrupted source code. Not a few developers may think that the continuous integration is only about compiling a source code and launching tests. In the next sections, we will present the steps of the continuous integration and describe these individual phases in detail.

To imagine the process, there is a illustration about the components and their connections in the process of continuous integration in Figure 2.1. The image illustrates situation when *Developer 1* commits changes to the version control repository. The CI server detects this change and provide a feedback about the change back to the *Developer 1*. The *Devel-*

oper 1 can review informations about change that he made in the given feedback, e.g. tests results.

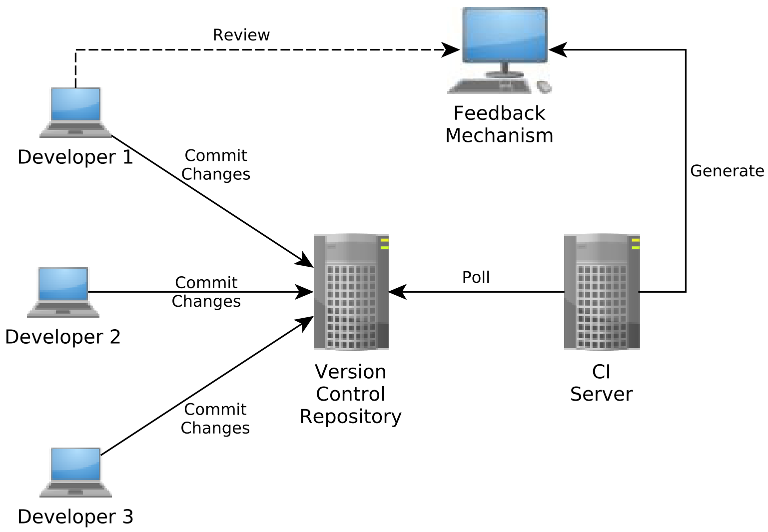

Figure 2.1: Components of continuous integration system [27].

2.2 Demands of Continuous Integration

The minimal requirements for a good software development of a project where multiple developers are working on the same project are a version control repository and a continuous integration server. The version control system guarantees a software configuration management which is required for the continuous integration. The meaning of the version control system is very important. You cannot manage changes that developers had made in the source code without a version control system. The version control system has a very positive impact on the developing project. The system offers a history of changes which may be highly useful if a rollback is desired. Besides the history of changes, this system may save more other information about the source code, e.g. who did the change, when was the change created, etc. In addition, the version control system represents a primary source for the project source codes. This type of project setup is much more often used these days than in the past. Nearly every project has its own version control system which is provided by a repository hosting service.

A CI server has a huge advantage. This is a reason why it is highly recommended. It depends on the developer, how does he deploy the CI server. With the CI server, he does not have to bother with such many scripts for the automation. Nevertheless, as he decides how the CI server will be established, the system must contain these features. To facilitate the process of continuous integration, the system must support services as polling version control system, retention of build history, launching predefined steps such as scripts and tests. Furthermore, the system should offer an opportunity to send a feedback back to the developers. This server executes a series of actions or steps taken in order to achieve a particular end of CI. The next section will determine and state these fundamental steps of the continuous integration scenario and describe and illustrate them in detail.

2.3 Stages of Continuous Integration

The stages of CI insure code inspection and code integration. Before we begin, we need to clarify certain concepts which will be used later. To understand these steps, we need to understand what is the difference between **a build**, **a private build** and **an integration build**.

Definition 1 *A build may refer to a set of activities performed to generate, test, inspect, and deploy software [27].*

Definition 2 *A private build define a process in which a software developer runs the build on his local machine to ensure that the changes he made work before he commits them into a version control repository.*

Definition 3 *An integration build is the act of combining software components (programs and files) into a software system [27].*

Figure 2.2 mentions the before stated Definition 3 which depicts the result of combination individual parts (components) of the software into a single software system. The transformation process that integrate these software components together into a one unified entity is called as an integration build.

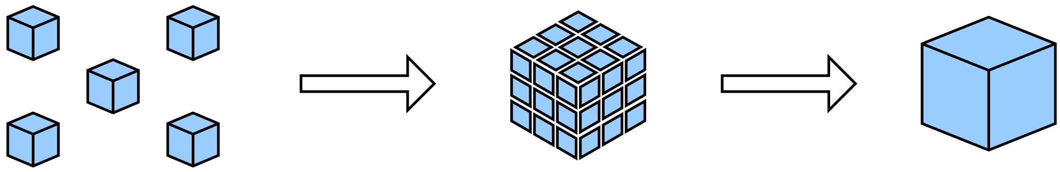

Figure 2.2: Integration build.

Now as we know what are these concepts we will illustrate the basic stages of continuous integration. To describe it properly, imagine that we have a group of developers working on the same project using a version control system where the source code of the software product is held, and they use a continuous integration service. The stages of continuous integration are the following:

1. **The change**
   One developer who wish to make a change, adjustment, improvement or to create a new feature in the software product has to clone the remote version control repository to his local computer to download the source code of the software product. At this point, he has a local version control repository in which he will do the changes he would like to. After a change is made, the change is only in a local repository and the developer would like to commit it into the remote repository. Before publishing the change, he has to run a private build. The developer has to publish the change he made which is a request for an approval of the change ready to merge into a specific branch on the remote repository. These not merged changes are published on the remote version of the control repository.
   By committing changes to the version control repository a continuous integration server is invoked. The continuous integration server polles the version control repository when a change is detected, after this poll a reaction occures.

2. **The reaction**
   When a change is detected it invokes a continuous integration server to execute a few tasks. The tasks are predefined in a build script which has to integrate the change with the rest of the source code of the software product. The script provide source code compilation, database integration, testing and code inspection. The execution of the script is referred to as an integration build.

   This stage of continuous integration usually includes also code verification. It finds defects or errors made by developer, e.g a compilation fail, tests failures etc. The errors are detected by tests which should have high code coverage. A number of errors in this stage can be reduced by launching a private build which may be less complex compared to launching the build script. Passing this stage depends on success of the build script which must be success on 100%.

3. **The feedback**
   The continuous integration server generates a feedback associated to the results of the build which is assigned to this change and it might be sent to the author of the change. There is log information generated every time, by passing the reaction stage, and it is held and assigned to the change. Feedback is given to the developer in a certain predefined form, e.g. email with failures only. The log file is saved on the continuous integration server where there is an overview about the builds and their stats.

4. **The waiting**
   This stage is the end of the process. It stands for continuous polling of the version control repository waiting for a new change. Detecting a change will cause launching the stages from the beginning.

2.4 Continuous Integration Server

If the software development proceed to use continuous integration in the workflow it might have a configured CI server. The principal sense of a continuous integration server is to get rid of a manual integration build. The configuration of the CI server depends on source code verification requirements and on a type of polling. The CI server can also provide an additional automation for necessary essentials to the development such as integration, deployment, etc.

The continuous integration system is based on automation that is conducted by CI server. Automation is an act, when manual tasks are united and executed together in order to simplify the execution of manual tasks. Nowadays, in software development automations can be found in different parts of software development. It helps to accelerate the development process. In a CI system, there are different types of builds and mechanisms used for the automation.

2.4.1 Polling

We can distinguish several types of build mechanisms such as on-demand, scheduled, poll for change and event-driven mechanism [27]. The simplest automated mechanism, on-demand mechanism, can be done by a single script and it helps to get rid of tasks repetition executed by the developer. The on-demand mechanism is an user-driven process in which someone manually initiates an integration build [27]. Scheduled mechanism is a planned event accomplished by a CI server in predefined time. In the situation, where multiple developers are frequently working on a product during the day, the best choice for a build should be to plan it in night. The scheduled type is used particularly when an advanced build of the software product is needed to be done. Scheduled processes are driven by time, for instance, so that it runs on an hourly basis, regardless of whether or not a change has occurred [27].

Poll for change mechanism and event-driven mechanism differ only in a way of invoking. Poll for change mechanism uses a periodic time for a change polling and the event-driven mechanism is time independent mechanism which is invoked by a version control repository. In a poll for change mechanism, a process wakes up in a regular intervals and checks for changes to the version control repository, if changes are detected an integration build is ran [27]. The event-driven mechanism is triggered by a version control repository. If change was detected by a version control repository then it initializes the build script. Only in these two mechanisms, there is a polling service which is sectionalized into two different types.

Types of polling can be divided into two parts - time dependent polling and change dependent polling. The CI server with time dependent polling is configured to check the version control repository for a new change in predefined periodic time intervals e.g. every 10 minutes. Contrawise, the CI server with change dependent polling is invoked with every single action which is a change in a version control repository via an informative message about the current action sent to the CI server. This message including event stats is triggered on a specified event in the version control repository which must support this feature.

Time dependent polling is mostly used in general due to inadequacies such as missing event triggering in the version control repository. Due to this fundamental feature some of CI servers has to have periodic polling on time. The main disadvantage is the time taken by downloading the actual source code from the repository. After the download is complete, the changes are still unknown, and so a comparison must be done between the latest and the last source code for the purpose of obtaining the new changes. Change dependent polling downloads only the real change towards the actual source code status made in the repository. If the version control system can support this feature the source code synchronization is much more faster and efficiently done.

2.5 Build Script

Instigation of CI system begin with a change in the version control system resulting in build script execution. Transforming sources into a system and simultaneously providing a review about the transformation is an intricate process also known as continuous integration, delivery and deployment. A CI system uses a build script allowing build automation, which includes every predestined statement to execute. This automation had a magnificent impact on software development. To get rid of constantly repeated actions for the purpose to accelerate the software development a build script was created. The principal script consists of a set of subscripts, which divide the automation into segments that are bound to themselves according to the execution order. Segments are shown in order in Figure

2.3. It shows the logical parts of a build script. Script performs a build also called as a software build which is not just about the source code compilation and tests launch. These various smoothly executed parts construct a functional unit of the software product. A working function unit congregation leads to working software deployment as the final step of CI. The script warrants simplification because of the developers adjust the source code and they are able to gain instant feedback about their work. As Martin Fowler said "Get everything you need into source control get it so that you can build the whole system with a single command." [20].

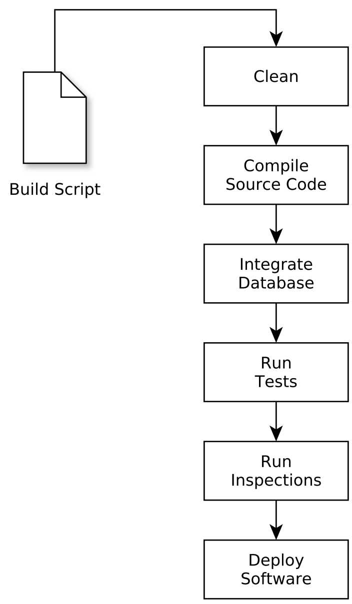

Figure 2.3: The logical processes of a build script [27].

The whole point of continuous integration is to provide a rapid feedback [20]. Developers would like to have as fast feedback as possible. To guarantee this quality there are different types of build scripts provided on different kinds of requests. Build scripts are divided by the role as lightweight and heavyweight scripts. Lightweight scripts are much more faster than heavyweight scripts. They are used on principle of speed. To ensure this behavior, at first the lightweight scripts are initiated because they can easily catch the basic vulnerabilities and then more advanced tests, inspections, and others are launched by heavyweight scripts which leads to an integration build. Martin Fowler marked the lightweight script which does the first build, as a "commit build" [20]. These scripts endeavor for quickness, error detection and software integration, besides that, they also provide a feedback about the results of the whole process to the developer.

A script is required due to build automation to provide a "press to build" functionality which is executed many times without any interaction. The script has its logical parts shown on Figure 2.3. The transformation process in the first part starts with a clean build, which is nothing more than just clean code compilation. The database integration and the tests execution may be executed differently because of dependence of tests from the database. Not every test depends on the database, due to this, we can divide them and we may run the database independent earlier than the dependent. If there is an error detectable by database independent than it is caught earlier what is more effective according to the time. After this phase, code inspection is launched for further deficiencies. The last stage of the build script is triggered after every previous stages ended successfully. Outcome of this, the build is an observable result with a log that reflects the build pipeline which forms the basis for the feedback generated for the developers.

2.6 Research about the Builds of Continuous Integration

Continuous integration is a practice, not a tool [10]. Martin Fowler on first of May 2006 stated the basics of CI and the best practices of CI in his article in which he remitted on still popularizing usage of CI. In addition to this article, there was a research provided by a group of scientists about the CI on project provided for the most part from GitHub. Their research is an empirical study about the usage, costs and benefits of CI which are concisely shown in abundant diagrams. The observation of CI and its usage pointed out the significant essential role of CI in open source projects. On the basis of the informations obtained from the researches about the CI, we can make a judgment that this practice will be more and more used in the open source projects. Thanks to automation and standardization, CI helps to effectively prevent errors when deploying applications into operation [28].

Continuous Integration is also referred to as a "cure for human error in deployment" [28] because of error prevention which is rapidly reduced by using this practice. The job of a developer includes a project build repetition which may be also reduced in the sense of tasks rate reduction applied on developer. These processes leverage extensive automation and encourage constant code sharing to fix defects early [17]. Many of errors, bugs, defects and vulnerabilities are reduced but not every of them is detected by using a CI, but nevertheless the manual software integration is excluded because of CI comprises it as the last step of the software deployment. The impact of the CI usage in software engineering will have extreme influence on the future of IT, more precisely in agile teams using extreme programming technique or any other agile technique. The usage of CI is very adaptive and versatile and it will be more and more used in forthcoming open source projects or any another projects which may not be open source only.

In general, if any group of developers would like to use a CI practice, they should fulfill few standards and take heed to these standards. In order for developers to benefit from use CI in practice, they should change their typical day-to-day software development habits [29]. Usage of CI is a beneficial sideline when a developer commits frequently, daily, often, probably few times per day. Farthest, the project should be hosted somewhere on any kind of version control repository which represents a main source for the source code of the product. Besides these two sole development requirements, the expectations are that the developers should not try to commit a broken code. It is avoidable by initiating a private build on their local machine, which decreases the fail chance of the build launched by the CI server.

By using CI practice, the risks as software corruption and integration problems are reduced appreciably and any kind of bugs are uncovered quickly. The integration may take unpredictable long time but the use of the CI practice resolve this problem by integrating the software frequently which may result in a few small kind of integration issues. Some other software methodologies integrate their work once after a long time which brings their software to face an incredibly huge integration problem. Martin Fowler pointed this problem in his article: “I was told that this project had been in development for a couple of years and was currently integrating, and had been integrating for several months.” [20]. Several articles describe this long time integration as a *Big Bang Integration* [11]. As we can see on Figure 2.4 the risk of the software integration is markedly reduced by using a daily (continuous) integration which is used in a CI practice.

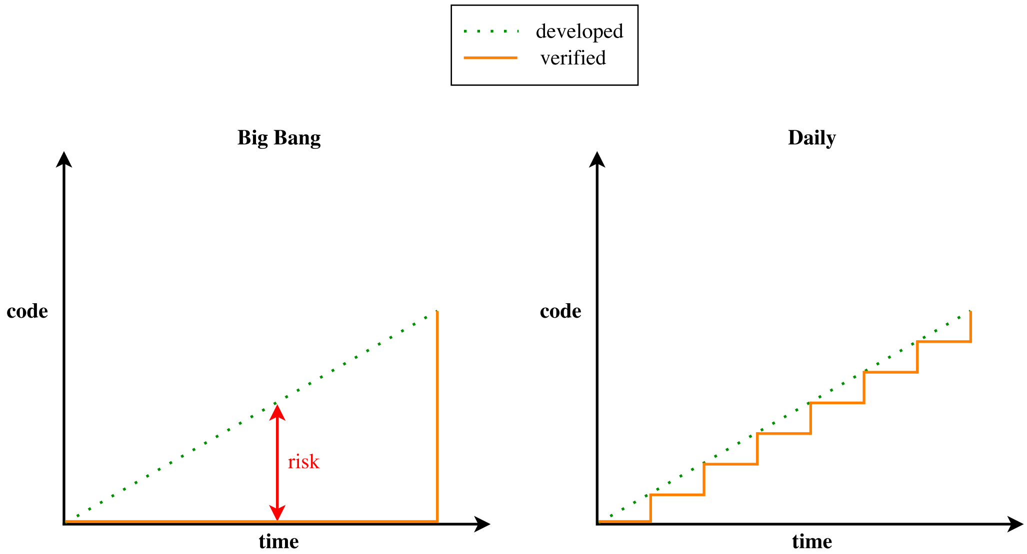

Figure 2.4: Comparison of integration builds [11].

> *“Not integrating continuously is expensive. If you don't follow a continuous approach, you'll have longer periods between integrations. This makes it exponentially more difficult to find and fix problems. Such integration problems can easily knock a project off-schedule, or cause it to fail altogether.”*
>
> – ThoughtWorks® [30]

Prevention against any type of error in a CI is solved via integration build performed on a CI server. Predicting the result of build has drawn the interest of academia and industry [15]. In term of build result analysis and prediction, most existing studies focused mainly on a large software project developed and maintained by big companies [15]. Travis CI community has created a TravisTorrent[1] [24] for the purpose of providing a huge amount of information about the builds for full-stack research on continuous integration which is still a developed prototype. Alongside the TravisTorrent, the GitHub company has provided information about the data inside of their version control system in a project called “The GHTorrent project”[2] [14]. Based on these given informations as an open dataset, a few analysis were conducted on them resolving the CI practice in a real life developed projects. The best practices were established on the results of the empirical studies of these obtained data sets provided by many of associations.

Studies about the CI were facing against difficult analysis due to inaccessibility of the project's data such as projects of private companies. Due to this, the observations are

[1] The name of TravisTorrent was chosen to resemble the close proximity to the GHTorrent project [4].
[2] The name signifies a torrent of data coming from GitHub [1].

related to projects mainly hosted on GitHub and predominantly using Travis CI. Informations received from these observations are impressive and they point to the popularizing usage of the CI or promoting CI adoption in projects nowadays. Continuous integration is emerging as one of the biggest success stories in automated software engineering [22].

The most interesting question is that how many projects are using the CI at all? In year 2016, an empirical study stated 40% [22] of the projects, observed by them, are using CI practice. Despite of this result, they noted that the number is still growing and it will be still growing more intensively. While CI is widely used in practice nowadays, we predict that in the future, CI adoption rates will increase even further [22]. The results are shown in the Table 2.1.

Table 2.1: Usage of CI in projects [22].

Projects Uses CI?	Percentage	Number of Projects
Yes	40.27%	13,910
No	59.37%	20,634

The adoption of the CI to the project may depend on many of factors which includes, for example, familiarity developers with the CI which is the main factor. The median time for CI adoption is one year [22]. The basic reason of putting the CI into the development is due to bugs and error reduction, but there is still a possibility to a bug or error incursion into production. As Martin Folwer said "Continuous Integrations does not get rid of bugs, but it does make them dramatically easier to find and remove." [20].

2.7 Best Practices of Continuous Integration

The continuous integration practice has become very exploited and its usage has increased considerably the overall agility and efficiency in the development process. It helps stakeholders, testers and product owners to work together seamlessly eliminating bottlenecks and achieve faster time to market [7]. This section describes fundamental practices which can lead to dramatical decrease of the costs on the project by using this approach to the CI practices. The costs reduction may be approximately 40% less as the Ade Miller's study [6] has shown. The influence may be avowedly known while using these practices. The effort of maintaining the CI system and the usage of the fundamental practices for CI has a magnificent impact on the project. Project investment into a CI can expect to achieve costs reduction, agility growth and error reduction in the development process. Some of the scientific studies report a different count of the CI practices but the general idea of these practices is same in everyone of them. Studies investigate more likely GitHub projects due to the free available informations, from GitHub projects in the GHTorrent and TravisTorrent projects, and on this basis they established these practices.

2.7.1 Maintain a Central Code Repository

As a fundamental requirement for the CI is a version control system where a principal repository is held. Software development project involves multiple developers constantly working and pushing code files that need to be orchestrated together to build a product [7]. Maintaining a system like this includes a lot of advantages as a source code backup, a reference on the primary mainline of the code with the latest content and a much more others. However, it is mainly used as a source for latest and clean source code of the developed project. This is a basic part of the setup for every project developed by a group of people who would like to share a code in the most common way in the development life. Nowadays, it is a often used practice nearly for everybody working on some software product in the development and alongside CI practice is necessarily used.

2.7.2 Commit Code Frequently

The continuous integration is used in agile software development where a huge count of changes are created and quickly integrated into the software product. From this, we can assume, that it points to a frequent code changing. How should we make a recommendation about this to the developers? The essential idea of this practice is to lead the developers to the thought in which they do a little logically compact change which will be committed as fast as they can to the version control repository. A commit should have a characteristic attribute of atomicity which divides the code in two parts as before adjustment and after adjustment. Many of developers do not practice this positive hint of their work. Developers should not wait with the committing of their work after finishing a task given to them. They should commit at least once per day into the baseline and commit a logically comprehensive unit of changes. Version control repositories are based on the source code of the project sharing between developers to accelerate the development process and the developed changes inside of this invokes a CI system which will automatically integrate the change into the software product. A conclusion of this part as a recommendation to the developers may sound like "commit frequently logically comprehensive commits".

2.7.3 Do not Commit a Broken Code

Broken code is a code that contains any type of failure when it is included in a CI build [29]. As a prevention against committing a broken code to the shared code repository there is an opportunity to run a private build on a developer's machine before each commit. A private build detects the the simplest mistakes such as a syntactic error or any other error forgotten in the code which are easy to detect. To reduce the plain error inside of the committing code, developer may use a code linter, if a developer uses an IDE[3], it may has a build in the linter. Code linting is a process of running a program which provide a code analysis for potential errors. Some of these basic errors are not detected by the linter so this is the reason why a developer should launch a private build before every single change he would like to commit. A private build may include a simple test set to detect these potential errors and defects in a created change. To commit a non broken code stands for to run a private build on a developer's local machine before committing his changes. This circumvention may accelerate the change integration.

2.7.4 Fix Broken Builds Immediately

An error consequences may beget a broken build as a result of the CI failure. Outcome of this action is an feedback which is sent to the developer as a fix requisition. Developer responsible for this problem should fix it as fast as possible irrespective of the build time cost. Fixing a broken build should be the top priority of the project [29]. As Martin Fowler quoted Kent Beck in his article "nobody has a higher priority task than fixing the build"

[3]Integrated Development Environment

[20]. The meaning of this build fixing is not to stop the actual tasks given to the developers, instead of this, it means to get couple of versed project members to fix the build. CI is effective while the build of the mainline, the principal branch of the project, has a successful termination result. Keeping an operational code in the repository forms the basis of CI which signifies a development on a stable based code. Effectiveness of the CI, while the code is stopped at build and not progressing to the integration into the software product, is none. To keep a mainline without broken build is almost always an unfeasible or nearly impossible task due to the human factor. The core idea of this part, how to ensure a mainline without broken builds, is to prioritize the urgent fixes and realize them if needed.

Some of the build fix solutions involve dropping the last commit - last change reversion. To avoid broken builds and enhance the solution mentioned before a new practice has been introduced - the pending head. Usage of pending head is a prevention for broken builds of the mainline. A pending head is a way how to indirectly commit a change into the mainline for the reason of a build make. The result success of the build decides about passing the commit into the mainline.

2.7.5 Keep the Build Fast

The stumbling block of the CI practice is the duration time of build. Because the build and test steps must be performed frequently, it is essential that these processes are streamlined to minimize the time spent on these steps [17]. Build time may be inappropriately long, which is unacceptable for the developers and can lead to the dysfunction of CI. Every minute you reduce off the build time is a minute saved for each developer every time they commit [20]. The most crucial and meaningful solution for reduction of the build duration is inside of the build pipeline. Build is executed by a build script which can be divided into parts which were described in section 2.5. According to the reduction of the build time, tests take a long enough part of the build time too. A CI practice laboriously rely on the unit test which runs approximately equal to core components which makes them very fast. They are the first line of defense in ensuring quality [9]. Vice versa, the API[4] and functional tests are greater time consumers due to their complexity. Graph in Figure 2.5 depicts the dependence of these two factors. A solution of this situation lies in a well chosen pipeline of the deployment, more precisely, how to run multiple builds in phases properly. When possible, running different sections of the test suite in parallel can help move the build through the pipeline faster [17]. A useful consideration of using many of unit tests with high code coverage which have minimal maintenance may also lead the build to time reduction.

[4] Application programming interface

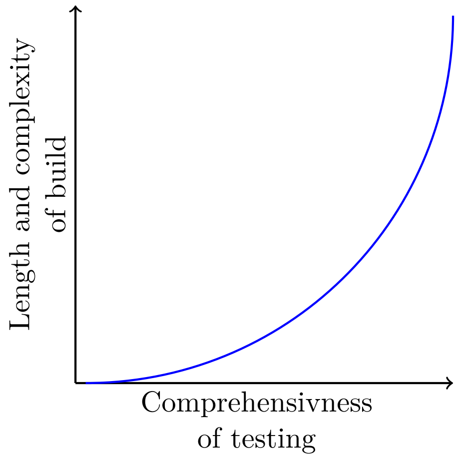

Figure 2.5: Dependence between length and complexity of the build and comprehensiveness of tests [3].

2.7.6 Everyone Can See What is Happening

Continuous Integration is all about communication [20]. Using a CI practice means to share all the gathered informations with the project members. Anybody from the team members should see the informations about the adjustment in the code that somebody had created. But it is not just about others work it is about the project state and the changes which have been made inside of it and about the new ones which will be integrated into it.

The fundamental part of the CI is the granted feedback about the result of the build realized by the CI server. Feedback is a summary of the log generated during build. These informations about the build status should be easy to obtain for anybody ensuring the development speed and quality. Developers must know the status of their adjustment after being handed over to the build. The news in the feedback are very important, especially at some build break. Information obtained from the feedback serves to fine-tune the made adjustment by the creator. Every single build result is assigned to the belonging commit (adjustment) which was made. These informations should be retained for the case if somebody would like to look for a build passing in the past - build passing before the current state. Developers should easily gain these informations and they should be notified if any kind of build broke arises on their work.

2.7.7 Automate Deployment

Automating deployment helps to reduce waste [10]. Automated deployment is nearly adherent to release automation. An essential part of releasing a software product is deploying it, first on development environments, then on QA[5] and UAT[6] environments, and finally on the real production environment, either on the developing organization's premises, on a customer's premises or on the cloud [3]. The usage of the CI required multiple development environments. It comes to this, that you have to move the binaries between multiple environments what follows to create scripts if no manual work is wanted. This allows to deploy

[5]Quality Assurance
[6]User Acceptance Testing

application across various heterogeneous environments used in the development process including the final production environment automatedly. In these days, there is an interest in virtualization which allows to create the expected environments easy and simple putting together these virtualized environments.

If the application meets all standards and criteria it is deployable. Deploying the application into the production signify carefulness you have to pay special attention to. There always was, is and still will be a chance of a failure, due to this fact a failure of application deployment requires a rollback. This rollback provides a certain decrease of difficulties about the deployment. Automated deployment, tied into good CI discipline, is essential to make this work [20].

Chapter 3

Automated Code Review

Development has adopted code review practice a long time ago. As software engineers collaboratively develop software, they need to understand, analyze, and validate past and present program modifications made by other developers in order to detect inconsistent, potential defects, manage the impact of the changes on structural anti-patterns, and avoid validation failures due to lack of test coverage [32]. Nowadays, it is an ordinary well-known practice which influences the overall code quality. Reviewing the source code is a complement to other quality mechanisms, such as compiling, integrating and testing [16]. This practice rests in idea of revising others work by others which points to collaboration on the same code by multiple people, especially co-workers. Today, this practice has been influenced by a lot of development practices and habits which lead the code review into a development progress. An idea has arisen to automate this process which should speed it up due to its time costs. The practice has been successfully automated but it is not fine-tuned already. There are many suggestions how to improve the code review quality and speed which cause code review more effective. The next sections describe code review, its types and how it is deployed in today's software development.

3.1 Code Review

Code review is a substantial part of the development which improve the source code quality markedly. The importance of the code review lies in the code enhancement that is significant towards not reviewed code. Code quality is made by imperfection reduction. Analysis of the code by someone else than the author who has a different type of view on the code may result in an imperfection detection. The reviewer, who is not the author of the code, can be a person or a software. This reviewer type division involves two types of code reviewing - the automated code review and the non automated review which is done by a developer. Human is irreplaceable by a machine but machines do not do mistakes. This is the reason why the development process includes the both type of code review. Automated code review functionality is supported in many of IDEs which informs the developer about the vulnerability in real time. This feature may involve static code analyzing tools which provide an extremely fast feedback. Rigby and Bird (2013) find that current software inspection practices tend to converge on Modern Code Review (MCR) [26]. The non automated code review also known as manual code review done by a person via some code review tool. The person is usually a project member who has to known at least the fundamentals about the code which he has to review. Review ends with a code criticism which should be taken by an author positively because of it enhances his code not degrades it. Software code review is a well-established software quality practice [26]. Code review can improve the quality of software products by identifying weaknesses in changes early in the development cycle (Fagan 1999; Shull et al. 2002) [26].

3.2 Principle of Code Review

The resulting work of a person normally involves deficiencies and faults because of the imperfection of people. The amount of imperfections depends on the skills and the experiences of an individual but they are still not removed completely. To catch the rests of non-caught vulnerabilities require to examine the work, the source code in case of development, by collaborators or project members. The examination of the work result is called in the development as a code review. To understand what is a code review or source code review there are different type of definitions.

Definition 4 *Source code review is an act of consciously examining source code intended to find bugs at an early stage of software development [18].*

Definition 5 *Source code review is an offline task aimed at finding the bugs in a code without compiling or executing the code [18].*

Usage of this practice is reflected on the quality of the source code which is rationally premeditated due to different types of reviewers view. Code review explicitly addresses the quality of contributions before they are integrated into project's code base [25]. A research article stated that a large portion of faults has been found by only one reviewer [21]. As many of reviewers are participate on a review, many deficiencies of the code are annihilated.

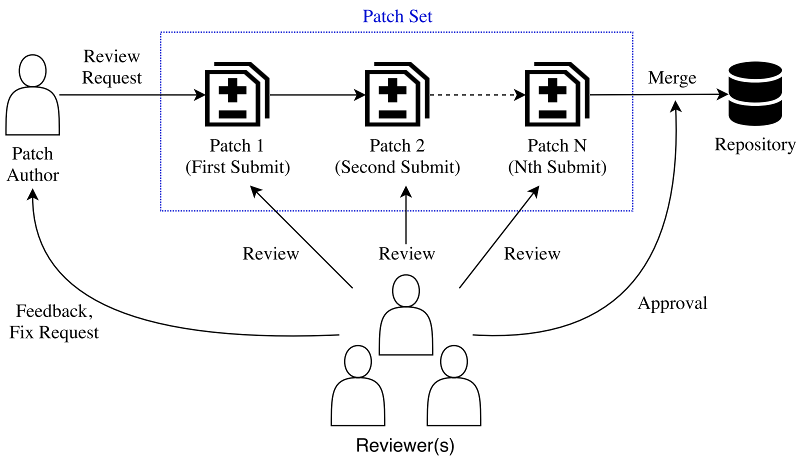

Figure 3.1: Code review process [31].

Code review always starts with a request for a review of some patch which is a modification of the actual source code. This patch is reviewed by somebody who has knowledge about this field or it is related to his field. The reviewer may approve this patch which leads to a merge of the patch into the actual source code in the repository - the project code base, or he may request for a fix from the author. This request for a fix does not mean that the patch is impaired, it will enhance the patch instead of patch degradation. After a fix request, the process is repeated until an approval. This process in non-automated, also called manual, because it is done by a person or a group of people. In an agile development, a thought has arisen which tried to automate this process. The impact of continuous integration on code review process is not yet properly understood given that they are interleaving steps in the software quality management [23].

3.3 Types of Code Review

There are many of code review types depending on the aspect, view and the review provider. Code review is divided into two basic types such as manual or automated. The reason of this division rests in the reviewer type. The reviewer may be a person who has to review the whole code or a software which processes the code according to the predefined set of rules.

The manual code review is a code examination on others work provided by a person. This type of review was described in section 3.2 and shown in Figure 3.1 what constitutes the basis of this practice. The difference to automated code review is the fact, that the manual code review includes a human person who has to give the review judgment - the approval or a request for a change.

To automate this practice, it is necessary to have a definition of project-specific rules. The automated code review is based on a predefined set of rules and best practices which are checked by a software whether the conditions are met or not. Matching these rules is provided via software which includes static analysis tools for this operation. As manual code review includes at least one person, the automated code review includes a static analysis tool which represents the person an performs his job in the code review. Static analysis tools for automated code review are the most effective [13]. This automation is only refinement of manual code review due to its duration.

Nowadays developments usually make use of both approaches. The major cause is to catch as many deficiencies as possible to reduce the insufficiency of the software product and increase its overall quality. Manual review is such a pain that reviewers regularly suffer from the “get done, go home” phenomenon - starting strong and ending with a sputter [13]. This factor may be circumvented with automated code review but it has its deficiency too because it is limited by the rules count. It cannot catch defects which are not defined the the set of rules. These two fundamental reasons lead the development to adopt both types of code review.

3.4 Automated Code Review

Automation cannot be achieved without any static analysis tool. A static analysis tool is included in a software which provide a static analysis which is performed without any compilation or any execution of the analyzed source code. Static code analysis finds a wide range of issues such as code style, code best practices, security, complexity, compatibility etc [8]. The terminology includes an expression defined for a static code analysis which is named as a code inspection. The software which provide the static analysis of the source code with the adequate tool mentioned before is called as a linter. Also the usage of this software has created a notion “linting” which is a process of running this software that analyses the source code. There are many linter types which variety depends on the language of the source code it has to analyses for the deficiencies. These widgets are often used in a CI practice due to error uncover before applying the modification and adjustments into the source code.

Automated code review is a process in which a software checks the source code for compliance and observance predefined via rules and for insufficiencies which could lead to potential errors. These rules represents a specific patterns which have to be adhered. Keeping the rules leads to better code quality and orientation of developers in the code due to one coding style which was chosen to be abode. This type of code review is an analytical solution for code checking which does not include source code compilation. The result of this process involve a list of violations and contravention of standards and principles which have not been complied with. The source code after solving these detected inaccuracies is faultless and without any potential error and also in one coding style. Due to the fact that this type of code review force developers to use only one approach to coding guideline which helps to make the code readability much more better unlike mixing multiple coding styles together as a result of collaboration. In agile software development with manual code review this practice has a considerable demand on the review speed in the development process. Because of this, oftentimes the development chooses a selection of both review types usage in the development considering that these types complement each other. Nowadays there are plenty of these application providing static code analysis which have their utilization nearly in every project.

3.5 Automated Code Review in Continuous Integration

This type of code review is oftentimes used besides a CI practice in which it has an appreciable impact. Many times the detected offenses break a build initialization in a CI pipeline. The build is useless if any of critical defect is included in the source code because of the time costs of the process which will even thought find this defect. Automated code review is fast enough to be used beforehand build initialization to decrease a chance of worthless build execution. Besides the usefulness of automated compilation and testing software projects can greatly benefit of the execution of automated static code analysis tools within CI [12].

Chapter 4

Open Source Projects

TODO

4.1 Open Source

TODO

4.2 Red Hat, Inc

TODO

4.3 Github

TODO

4.4 TravisCI

TODO

4.5 ManageIQ

TODO https://lwn.net/Articles/680060/

Chapter 5

Implementation

The fundamentals of the implementation part of this thesis were established on deficiencies of the ManageIQ Bot [5]. A list of missing needs of the bot were created by the developers and more important of them were chosen to be implemented. The goal of the tasks was to reduce these deficiencies by implementing and adding them into the currently working bot. Besides of the implementations of bot's missing features there were a plenty of problems and issues as a side effect which had to be resolved as fast as possible. The result of the features may help to the developers with a little increase of the team agility and simplifying some of their needs targeted on the bot. I hope and believe that these added features will help to the developers in their daily work. Some of the features are added yet, but some of them are still not added yet because they are waiting for their pull request merge or approval.

5.1 ManageIQ Bot

The ManageIQ bot is the ManageIQ team's helper to automate various developer problems [5]. The automation of these manually provided tasks by developers increased the team agility because of time saving for major tasks. In addition, this bot reacts on specific commands used by the project members on which he perform a desired action. The bot's core is based on Sidekiq[1] which is a simple background processing for ruby. Due to the fact that the bot is not using GitHub's Webhooks[2] he is configured to use a polling method instead. The polling method rests in a specific content downloading repetitively via GitHub's REST API v3[3] over HTTPS. The downloaded content contains JSON data from which are the necessary informations extracted and on their basis an action or multiple actions are performed. Some of the bot's actions results in a GitHub actions such as posting a comment, adding a label or milestone that are using HTTP requests via the mentioned REST API v3. These actions are used to facilitate the manual tasks resulting in a complete elimination of them from the developer's tasks.

[1] https://github.com/mperham/sidekiq
[2] https://developer.github.com/webhooks
[3] https://developer.github.com/v3

5.2 Pronto

Pronto is a tool that provides an automated code review over new changes in a git branch [19]. It is typically used in continuous integration as a way to provide feedback on a pull/merge request [19].

The Pronto [2] integration was necessary due to the unification of the output of multiple static code review services. Pronto provide a quick automated code review by analysis of the relevant changes with the related static analysis tool to the source code. The main advantage is that it has its own formatters which are very useful. If the output is desired to be formatted for GitHub, BitBucket, GitLab or any other supported format than the built in formatter provide this expected feature. The format type - pull request, pull request review, etc., is also supported. The result of formatters is configurable via configuration file which define the desired format of the produced message. Henceforth, Pronto is able to run multiple different static code analyzing tools which are united under Pronto. They are united because of only one output is expected with a summary of the analysis results.

To unite these various tools, Pronto had encapsulated every single tool into a Pronto plugin - a Pronto runner. This runner represents a middle layer between the Pronto application and the tool - the static code analyzing tool. The encapsulation facilitates the unification of these tools at the end of the source code analysis process. The pronto runner provide an automated code review for a specific programming language by the corresponding linter. This linter union was a great simplification of running all expected linters together.

5.2.1 Issues

During the integration process, a few of problems were discovered which had to be resolved. The main issue of this integration was a pattern matching of the pronto output with the original output of the code review. This integration required to create exactly the same data structure from the pronto result as the original data structure was because of further use in the bot - compatibility with the original structure of the results. Furthermore, a bug has been found during the testing of pronto. A pronto runner does not include an offense about a syntax error. This type of offense was detected by linter but the encapsulation of this linter into the runner caused that this offense was threw away.

5.2.2 Integration

The integration process of Pronto involves implementation of pronto result converter, unit tests, bug fixes and enhancements with some of them being designed after review of the pull request by the developers. To integrate pronto without changing the original behavior of the bot required to launch pronto with the authentic linters and convert the pronto result to match the original data structure pattern of the offenses. Obtainment part of the pronto creates a temporary folder in which is the entire folder including the examined repository copied. After that, a repository object is created based on the actual temporary folder content which is fetched. The last stage of this part rests in gathering the patches which are passed to the pronto runners. The execution of pronto runners results in the expected output - an array of pronto-message objects. An one object which has type of pronto-message contains every information about one patch of line including the linter result describing this line.

TODO
Related pull request: https://github.com/ManageIQ/miq_bot/pull/406.
Expected linters have to be specified only in gemfile by using the corresponding gems for the pronto runners responsible for the linters.

5.3 Pull Request Review Commands

TODO
Related pull request (ADD reviewer): https://github.com/ManageIQ/miq_bot/pull/408.
Related pull request (RM. reviewer): https://github.com/ManageIQ/miq_bot/pull/411.

5.4 GitHub Status API integration

TODO
Related pull request: https://github.com/ManageIQ/miq_bot/pull/412.

Chapter 6

Conclusion

TODO

Bibliography

[1] The GHTorrent project. [Online; Accessed: 2018-02-10].
URL: http://ghtorrent.org/

[2] Pronto. [Online; Accessed: 2018-02-13].
URL: https://github.com/prontolabs/pronto

[3] *Deployment Automation*. [Online; Accessed: 2018-02-23].
URL: http://electric-cloud.com/wiki/display/releasemanagement/Deployment+Automation

[4] TravisTorrent. [Online; Accessed: 2018-02-10].
URL: https://travistorrent.testroots.org/

[5] ManageIQ Bot. 2008. [Online; Accessed: 2017-11-02].
URL: https://github.com/ManageIQ/miq_bot

[6] Ade Miller: *A Hundred Days of Continuous Integration*. Aug 2008. doi:10.1109/Agile.2008.8. [Online; Accessed: 2018-02-14].
URL: http://ieeexplore.ieee.org/document/4599493/

[7] Anuradha Ishwaran: *8 Best Practices of Continuous Integration To Supercharge Your Software Development Team*. 2016. [Online; Accessed: 2018-02-14].
URL: http://www.tothenew.com/blog/8-best-practices-of-continuous-integration-to-supercharge-your-software-development-team/

[8] Codacy: *Automate Your Code Reviews with Static Code Analysis*. Feb 2016. [Online; Accessed: 2018-03-17].
URL: https://blog.codacy.com/automate-your-code-reviews-with-static-code-analysis-7d8ab0c81b03

[9] Dan Radigan: *Continuous integration, explained*. [Online; Accessed: 2018-02-23].
URL: https://www.atlassian.com/continuous-delivery/continuous-integration-intro

[10] Darryl Bowler: *Ten Best Practices for Continuous Integration*. 2012. [Online; Accessed: 2018-02-05].
URL: http://blogs.collab.net/devopsci/ten-best-practices-for-continuous-integration

[11] Eero Laukkanen: *Continuous Integration, Delivery and Deployment*. 2015. [Online; Accessed: 2018-02-10].
URL: https://mycourses.aalto.fi/pluginfile.php/161735/mod_folder/

content/0/T-76.5613_04-Continuous%20integration%20delivery%20and%20deployment_2015.pdf

[12] Fiorella Zampetti. Simone Scalabrino. Rocco Oliveto. et al.: *How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines*. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. May 2017. pp. 334–344. doi:10.1109/MSR.2017.2. [Online; Accessed: 2018-03-18].
URL: http://ieeexplore.ieee.org/document/7962383/

[13] Gary McGraw: *Automated Code Review Tools for Security*. *Computer*. vol. 41, no. 12. Dec 2008: pp. 108–111. ISSN 0018-9162. doi:10.1109/MC.2008.514. [Online; Accessed: 2018-03-09].
URL: http://ieeexplore.ieee.org/document/4712512/

[14] Georgios Gousios. Diomidis Spinellis: *GHTorrent: GitHub's Data from a Firehose*. June 2012. doi:10.1109/MSR.2012.6224294. [Online; Accessed: 2018-02-10].
URL: http://ieeexplore.ieee.org/document/6224294/

[15] Jing Xia. Yanhui Li: *Could We Predict the Result of a Continuous Integration Build? An Empirical Study*. 2017. doi:10.1109/QRS-C.2017.59. [Online; Accessed: 2018-02-10].
URL: http://ieeexplore.ieee.org/document/8004336/

[16] Jun-Suk Oh. Ho-Jin Choi: *A reflective practice of automated and manual code reviews for a studio project*. In *Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*. 2005. pp. 37–42. doi:10.1109/ICIS.2005.17. [Online; Accessed: 2018-03-09].
URL: http://ieeexplore.ieee.org/document/1515372/

[17] Justin Ellingwood: *An Introduction to Continuous Integration, Delivery, and Deployment*. 2017. [Online; Accessed: 2017-11-03].
URL: https://www.digitalocean.com/community/tutorials/an-introduction-to-continuous-integration-delivery-and-deployment

[18] K. R. Chandrika. J. Amudha. Sithu D. Sudarsan: *Recognizing eye tracking traits for source code review*. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept 2017. pp. 1–8. doi:10.1109/ETFA.2017.8247637. [Online; Accessed: 2018-03-03].
URL: http://ieeexplore.ieee.org/document/8247637/

[19] Kevin Jalbert: Create your own Pronto Runner. May 2017. [Online; Accessed: 2018-04-01].
URL: https://kevinjalbert.com/create-your-own-pronto-runner/

[20] Martin Fowler. Matthew Foemmel: *Continuous integration*. 2006. [Online; Accessed: 2017-11-02].
URL: https://martinfowler.com/articles/continuousIntegration.html

[21] Martin Höst. Conny Johansson: *Evaluation of code review methods through interviews and experimentation*. *Journal of Systems and Software*. vol. 52, no. 2. 2000: pp. 113 – 120. ISSN 0164-1212. doi:10.1016/S0164-1212(99)00137-5. [Online;

Accessed: 2018-03-03].
URL: http://www.sciencedirect.com/science/article/pii/S0164121299001375

[22] Michael Hilton. Timothy Tunnell. Kai Huang. et al.: *Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects*. 2016. doi:10.1145/2970276.2970358. [Online; Accessed: 2017-10-30].
URL: https://dl.acm.org/citation.cfm?doid=2970276.2970358

[23] Mohammad Masudur Rahman. Chanchal K. Roy: *Impact of Continuous Integration on Code Reviews*. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. May 2017. pp. 499–502. doi:10.1109/MSR.2017.39. [Online; Accessed: 2018-03-10].
URL: http://ieeexplore.ieee.org/document/7962406/

[24] Moritz Beller. Georgios Gousios. Andy Zaidman: *TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration*. 2017. doi:10.1109/MSR.2017.24. [Online; Accessed: 2018-02-10].
URL: http://ieeexplore.ieee.org/document/7962393/

[25] Oleksii Kononenko. Olga Baysal. Michael W. Godfrey: *Code Review Quality: How Developers See It*. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. May 2016. pp. 1028–1038. doi:10.1145/2884781.2884840. [Online; Accessed: 2018-03-03].
URL: http://ieeexplore.ieee.org/document/7886977/

[26] Patanamon Thongtanunam. Shane McIntosh. Ahmed E. Hassan. et al.: *Review participation in modern code review*. *Empirical Software Engineering*. vol. 22, no. 2. Apr 2017: pp. 768–817. ISSN 1573-7616. doi:10.1007/s10664-016-9452-6. [Online; Accessed: 2018-03-03].
URL: https://doi.org/10.1007/s10664-016-9452-6

[27] Paul M. Duvall. Steve Matyas. Andrew Glover: *Continuous Integration - Improving Software Quality and Reducing Risk*. Pearson Education, Inc.. 2007. ISBN 0-321-33638-0.

[28] Pavel Ducho: *Continuous Integration - Cure for Human Error in Deployment*. 2016. [Online; Accessed: 2017-11-03].
URL: http://www.web-integration.info/en/blog/continuous-integration-cure-for-human-error-in-deployment/

[29] Saba Hamdan. Suad Alramouni: *A Quality Framework for Software Continuous Integration*. 2015. [Online; Accessed: 2018-02-03].
URL: https://www.sciencedirect.com/science/article/pii/S2351978915002504

[30] ThoughtWorks: *Continuous integration*. 2017. [Online; Accessed: 2017-11-02].
URL: https://www.thoughtworks.com/continuous-integration

[31] Yuki Ueda. Akinori Ihara. Toshiki Hirao. et al.: *How is IF Statement Fixed Through Code Review? A Case Study of Qt Project*. In *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. Oct 2017. pp. 207–213. doi:10.1109/ISSREW.2017.32. [Online; Accessed: 2018-03-03].
URL: http://ieeexplore.ieee.org/document/8109285/

[32] Zhiyuan Chen: *Helping Mobile Software Code Reviewers: A Study of Bug Repair and Refactoring Patterns*. In *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. May 2016. pp. 34–35. doi:10.1109/MobileSoft.2016.026. [Online; Accessed: 2018-03-10]. URL: http://ieeexplore.ieee.org/document/7832964/

Appendix A

Appendix Chapter

TODO