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

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

Keywords

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

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

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adrián Tóth
February 23, 2018

Acknowledgements

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

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 and Automated Code Review

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 [11]. 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 subsections. 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 [8]. 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 subsections, 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 *Developer 1* can review informations about change that he made in the given feedback, e.g. tests results.

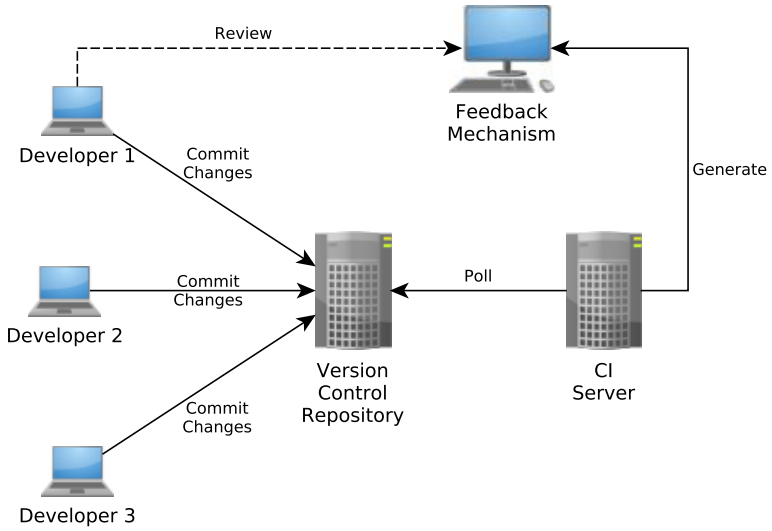

Figure 2.1: Components of continuous integration system [13].

2.1.1 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 subsection will determine and state these fundamental steps of the continuous integration scenario and describe and illustrate them in detail.

2.1.2 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 [13].*

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 [13].*

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.

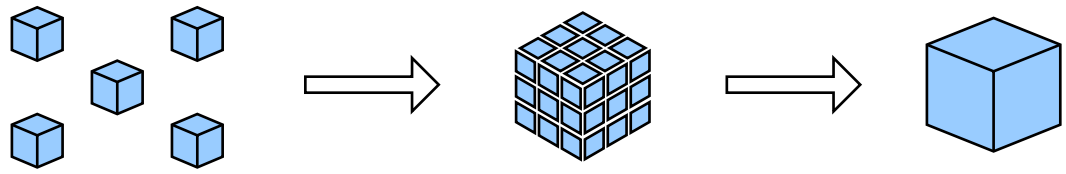

Figure 2.2: Integration build.

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.

2.1.3 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 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.

Polling

We can distinguish several types of build mechanisms such as on-demand, scheduled, poll for change and event-driven mechanism [13]. The simplest automated mechanism, on-demand mechanism, can be done by single script in order 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 [13]. Scheduled mechanism is a planned event accomplished by 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 a time planned 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, whether or not a change has occurred [13].

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 [13]. The event-driven mechanism is triggered by a version control repository, if change was detected by version control repository than 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, 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.

2.1.4 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 demand in 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." [10].

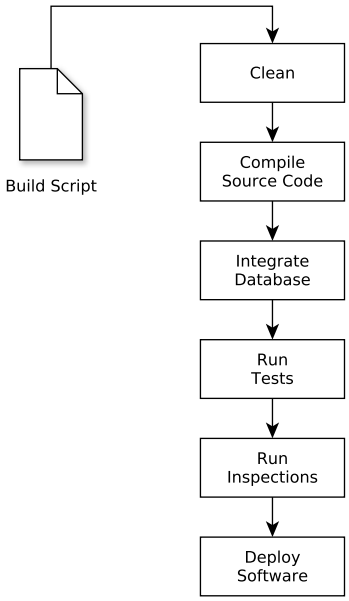

Figure 2.3: The logical processes of a build script [13].

The whole point of continuous integration is to provide rapid feedback [10]. Developers would like to have as fast feedback as possible. To guarantee this quality there are different types of build scripts provided on different roles 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" [10]. 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.

TODO: Build script and its parts description (shortly)

2.1.5 Research about the Builds of Continuous Integration

Continuous integration is a practice, not a tool [5]. 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 [14].

Continuous Integration is also referred to as a "cure for human error in deployment" [14] 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 to a few of them for the sense of tasks rate reduction applied on developer. These processes leverage extensive automation and encourage constant code sharing to fix defects early [9]. 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 demand in 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 dwell on these standards. In order for developers to benefit from implementing the practice of CI, they should change their typical day-to-day software development habits [15]. Usage of CI is beneficial sideline when a developer commits frequently, daily, often, probably few times per day. Farthest, the project should be hosted somewhere on any kid 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 decrease 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 kid 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." [10]. Several articles describe this long time integration as a *Big Bang Integration* [6]. 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.

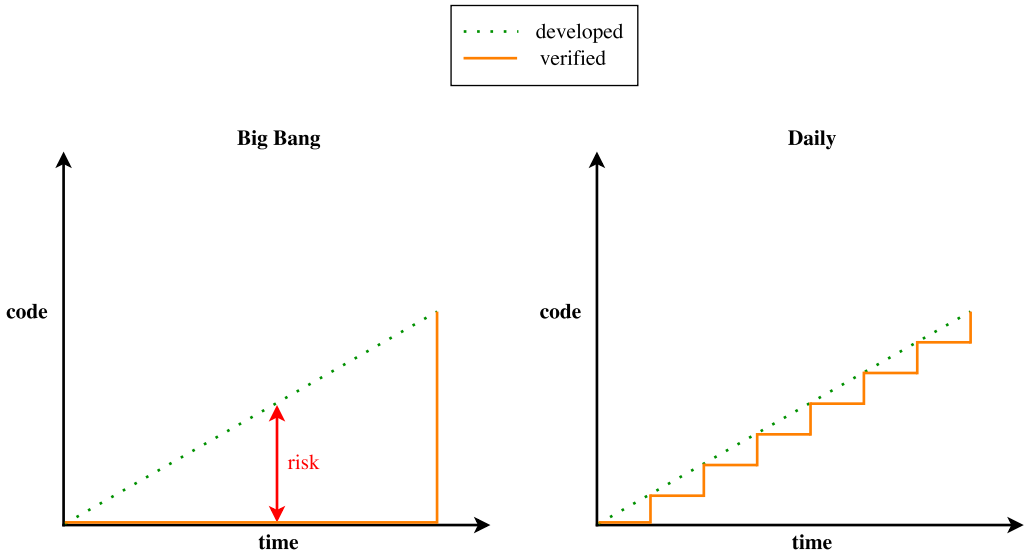

Figure 2.4: Comparison of integration builds [6].

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 [8]. In term of build result analysis and prediction, most existing studies focused mainly on large software project developed and maintained by big companies [8]. Travis CI community has created a TravisTorrent[1] [12] 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] [7]. 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.

TODO: Research diagrams / results

2.1.6 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 [4]. 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 [3] 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

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

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.

Maintain a Central Code Repository

As a fundamental requirement for the CI is a version control system where is a principal repository held. Software development project involves multiple developers constantly working and pushing code files that need to be orchestrated together to build a product [4]. 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 other. However, it is mainly used as a source for latest and clean source code of the developed project. This is basic part of the setup for every project developed by a group of people who wish like to share code with 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 it is necessarily used.

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 divide the code on 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”.

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 [15]. 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 one commit. A private build detects the the simplest mistakes such as syntactic error or any other error forgotten in the code which are easy to detect. To reduce the plain error inside of committing code, developer may use a code linter, if a developer is using an IDE[3], it may has a build in linter. Code linting is a process of running a program which provide a code analysis for potential errors. Some of these basics errors are not detected by 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 error and defects in a created change. To commit a non broken code stands for

[3]Integrated Development Environment

to run a private build on a developer's local machine before committing his changes. This circumvention may accelerate the change integration.

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 it is possible irrespective of the build time cost. Fixing a broken build should be the top priority of the project [15]. As Martin Fowler quoted Kent Beck in his article "nobody has a higher priority task than fixing the build" [10]. 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 signify 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.

Keep the Build Fast

The stumbling block of the CI practice is the duration time of build. Build time may be inappropriately long, what is unacceptable for the developers and can lead to dysfunction of CI. Every minute you reduce off the build time is a minute saved for each developer every time they commit [10]. 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 2.1.4. TODO...

TODO: List of best practices

2.2 Code Review

TODO: types(manual / automated), (+)(-), vulnerabilities

2.2.1 Automated Code Review

Asdf.

Chapter 3

Continuous Integration in Open Source Projects

Asdf.

3.1 Open Source

Asdf.

3.2 Github

Asdf.

3.3 TravisCI

Asdf.

Chapter 4

Conclusion

Asdf.

Bibliography

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

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

[3] 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/

[4] 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/

[5] 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

[6] 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

[7] 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://gousios.gr/pub/ghtorrent-githubs-data-from-a-firehose.pdf

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

[9] 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

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

[11] Michael Hilton. Timothy Tunnell. Kai Huang. et al.: *Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects*. [Online; Accessed: 2017-10-30]. URL: http://cope.eecs.oregonstate.edu/papers/OpenSourceCIUsage.pdf

[12] Moritz Beller. Georgios Gousios. Andy Zaidman: *TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration*. 2017. [Online; Accessed: 2018-02-10]. URL: http://www.st.ewi.tudelft.nl/~mbeller/publications/2017_beller_gousios_zaidman_travistorrent_synthesizing_travis_ci_and_github_for_full-stack_research_on_continuous_integration.pdf

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

[14] 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/

[15] 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

Appendix A

Asdf

Asdf.