Implantation of continuous integration practices: An experience report in a software development and research laboratory

Igor Muzetti Pereira[1], **Lucas Cedro de Lima**[1], **Vicente J. P. de Amorim**[1], **William S. Nunes**[1]

[1] Instituto de Ciências Exatas e Aplicadas
Universidade Federal de Ouro Preto (UFOP)
João Monlevade – MG – Brazil

{igormuzetti,lucaslima2004,vjpamorim,williamnunes11}@gmail.com

Abstract. *Continuous integration refers to the merge, build and testing stages of the software release process. Every committed changes should offer a return of your behavior to the team still in the development phase, implying in various benefits. This paper presents an experience report in implantation of the Continuous Integration in the development an Android application, seeking to analyze the behavior of the team and the impacts on the project. Answers of the applied questionnaires, the increase of the number of merges and other difficulties in implanting the practice in an research academic laboratory were discussed.*

1. Introduction

The intense condition about the software industry to be able to demonstrate results more quickly has revolutionized its development process. What in the past was synonymous with delays in delivery and increase in project costs, is currently a reference in methods that propose agility, allied to the quality of software. Within this segment of agile development, it is clear the importance of Continuous Integration, promoting a rapid response by incorporating changes in software, showing the team what works and what is pending, constituting also the real notion of completeness of a project [Fowler 2006], as well as leveraging the realization of other activities that make up the development process that can be "laid aside" during this accelerated pace [Ståhl and Bosch 2014a], preventing the team from waiting until the end of the project to integrate their modifications leading to probable software quality problems, with high repair costs and generating delays in project delivery [Duvall et al. 2007].

The environment of an academic laboratory seeks to develop ideas, conduct research, relate the theory obtained in the classroom with practice, in a certain way, also bring the student closer the professional environment of industry [Rodrigues and Estrela 2012]. That way, the mobile computing laboratory iMobilis[1], located at the Institute of Exact and Applied Sciences, campus of the Federal University of Ouro Preto, has implanted in its culture diverse practices of project management and teams, as well as the BOPE process [Pereira et al. 2013] and adaptations of Scrum and Extreme Programming (XP), as well as other software engineering practices, aiming at this approximation in addition to improving the management and quality of its projects. That way, providing also a continuous

[1] http://imobilis.ufop.br/

improvement of the process, so that, the expected final product be developed within the estimated costs and meeting [Rezende 2005], resulting in the success of the projects.

Among the projects carried out, highlights the MobMine research project, an application for the Android platform, which helps in the creation of solutions and automation of some internal processes of a multinational organization in the mining sector. This project has an initial duration of two years and has a considerable scope, therefore, its development is carried out by a team composed of ten members, relying on three guiding teachers and one financier partner of the research project, beyond the six developers who are largely students of the undergraduate courses of Systems of Information and Computer Engineering. Among them is present the role of an analyst of testing and a developer leader, which differently from the others, is not a student but a recent graduate researcher, responsible for performing the integration and analysis of increments of source code that are generated by other developers.

Given the progress of the project, the application goes into operation, in which some previously developed features begin to be, in fact, utilized. Since, new features and improvements are still in production, there is a great concern that these do not compromise the version that is in operation when they be integrated. Consequently, tests are performed to find the maximum number of possible problems that can be generated by introducing this increment. This process of produce, integrate, test, fix and release new versions is repeated while the application is under development.

In order to reduce the risks of software delivery, to obtain a rapid return on the behavior of these changes in the software and to improve the team's development methodology, the Continuous Integration was an interesting practice to be implemented by the team. Possibiliting in this way, treat in an improved way and constant the problems faced by the team in the act of integration, such as code conflicts and introduction of bugs in stable versions. Another important point is to minimize the effort and time spent to find and debug problems [Fowler 2006], transferring them to the development of new features, since most of the developers are students and are expected only fifteen hours of dedication per week.

This paper presents an experience report when reviewing the literature, design, propose and implement Continuous Integration practices in this project, obtaining good initial results observed by the team responses through a two-phase research, characterizing the scenario before and after the beginning of the implantation, besides the quantitative analysis of Merges carried out by the team in the project which demonstrated an evolution. The difficulties encountered during the implementation of the practice in an academic environment are discussed, characterizing a factor directly related to the challenges encountered, since most of the developers are students in be vocational training who are establishing their first contact with a real project.

The rest of the paper is organized as follows. The Chapter 2 presents the related works. The Chapter 3 presents the methodology and tools used in the development of implementation of Continuous Integration practice. The results are presented in Chapter 4. Finally, conclusions, challenges encountered and future perspectives are described in Chapter 5.

2. Related Work

There are several studies in the literature that portray cases of implantation and studies of Continuous Integration. These differ from each other in the tools used, project and team characteristics, interpretation, implementation process and even the results obtained [Ståhl and Bosch 2014b]. The study of the literature was carried out throughout the development of the work, with an initial stage of six months duration to find out what is, why and how to implement continuous integration. Search engines such as Google Scholar and the CAPES periodicals portal were used for research, conducted through keywords, initially with its term ”continuous integration”, and posteriorly relating it to others like ”agile methodology”, ”practices”, ”methodologies”, ”extreme programming”. The next step was to search for reports of cases with similar characteristics, thus, were related ”continuous integration”searches with the keywords ”android project”, ”academic environment”, ”implementation”. The relative works were selected considering their relevance, consequently those with publication in periodicals and conferences, besides the date of publication and points of intersection with this work, analyzing points of variation.

[Stolberg 2009] performed a report of learning and experience from the implementation of the practice of Continuous Integration in a team initially without any automation structure, counting on a manual build process. The sketch and the choice of it be Continuous Integration tools have been influenced by what would work and would fit the application development environment Windows .NET C #. From this experience, [Stolberg 2009] observed that tests in parallel with development can overcome the traditional disarticulation that normally occurs between both, beyond the be automation of acceptance tests intimidating at first, but attractive, worrying it only in relation to its scalability. The difficulty with the team was convince them to go through some changes in order to allow performing the Continuous Integration. [Stolberg 2009] indicates that in their next projects, a first step to make agile testing would be to consider the implementation of a Continuous Integration system to support them.

Majority of the Continuous Integration implementations are developed in the software industry consolidated environments involving high-skilled professionals. However, [Hembrink and Stenberg 2013] sought the academic environment, implementing the practice in one of several teams that developed the same student project, consisting of a simulation of a real Java project with agile characteristics. This team was trained by the authors throughout the project with the objective of developing the Continuous Integration, making adaptations in order to adjust the practice to the course environment, utilizing also by Jenkins[2] server. At the end of the project, it was observed by [Hembrink and Stenberg 2013] that among the teams, this was the only one that used an Continuous Integration server, automating the build and sharing the feedback. There was also a breakthrough in testing coverage and a low proportion of builds that failed, as well as increased the releases frequency, result of more manageable versions, and of the evolving code quality. Among the problems and challenges faced, [Hembrink and Stenberg 2013], lack of knowledge about the tests by the developers, and automation of acceptance tests, were more prominent.

Through a case study performed by [Hukkanen 2015] was investigated the adoption of Continuous Integration in an empirical scenery of a modular Java software project

[2]Continuous Integration Server - `https://jenkins.io/`

for telecommunications in Nokia Networks. Once the project is pretty big and stakeholders are geographically distributed, project size has complex made the effort of integration and test. Efficient use of practice has been hampered by problems related to testing, infrastructure, dependency management, communications, and practical aspects of Continuous Integration. These challenges interfere with each other and negatively impact productivity, which is related to build time and its results, debugging and even the Continuous Integration system. [Hukkanen 2015] also observed in his work that integrating each increment of code instantly may not be a desired state for all projects, and that immediate improvements may not appear when adopting a new Continuous Integration system. [Hukkanen 2015] also indicates that the way technical environments are configured and maintained seems to be crucial.

3. Methodology

Since the project is in full development and it is a real project,some care was taken when trying to implement the practice. Points such as the team's development culture, the tools used, and their testing practices were observed and treated in a way that their impact was previously measured and did not adversely affect the progress of the project.

The team defines the activities as of the user stories, proposed improvements by stakeholders, or from bug fix requests. Once defined and assigned to team members, these activities are developed in parallel and on several occasions in a single software module, making the development of this activity concurrent, besides be needed integrate the all modifications of the developers on the main software to generate a release.

This task of integrating the increment to the main software requires effort and time, because when encountering conflicts between the versions of the developers, the team leader performs an audit with the authors of the respective changes, comparing them and analyzing their consequences. At the end of the week, more specifically on Fridays, versions containing the changes that came up during the week are built. These versions are analyzed by the tester for bugs and inconsistencies. Throughout this integration process, bugs can arise and also go unnoticed by accepting or rejecting conflicts, which in turn are likely to only be discovered and reported together with the test analyst's results.

3.1. Planning

According to the characteristics of the project, research and comparative studies of tools that best suit the development environment of an Android application were carried out. Even the tools that were already used by the team like Android Studio[3], Git[4] and Gradle[5], were compared with other tools of the same nature as the IDE Eclipse[6], the version control system Subversion[7] and the Ant[8]. However, in several characteristics, the tools used currently have proved to be optimal, in addition to maintaining a good harmony of functioning and knowledge by the team. Android Studio, for example, is currently the

[3]Integrated Development Environment (IDE) for Google's Android operating system - `https://developer.android.com/studio/index.html`

[4]Version Control System - `https://git-scm.com/`

[5]Open Source Build Automation System - `https://gradle.org/`

[6]Integrated Development Environment (IDE) - `https://eclipse.org/`

[7]Version Control System - `https://subversion.apache.org/`

[8]Build Automation System - `http://ant.apache.org/`

official IDE for Android development and natively uses Gradle to perform the application build.

As Continuous Integration server was chosen the Jenkins after comparisons with other servers such as CruiseControl[9], Travis CI[10], TeamCity[11]. Was considered characteristics as popularity, placing Jenkins as the server used by almost two in three respondents [Maple and Shelajev 2016], have an open source license, compatibility with Git and the Android environment, and have great extensibility [Polkhovskiy 2016], which provides add plugins to use, for example, the Gradle build automation tool.

3.2. Implantation

From the definition of the tools an integration simulation environment was constructed for a fictitious project, using features present in the real project. This simulated environment had the objective of obtaining a first contact with the Jenkins, the concepts of the Continuous Integration and analyze the operation of this system.

After learning about the previous context of the methodology used by the team, an analysis was made and some improvements were proposed to the team regarding their development methodology, considering the concepts and principles of Continuous Integration presented by [Fowler 2006]. In some points the team showed up to get a certain advantage by already employing them, even if in some cases not correctly or in the best way, but that eventually made the abstraction by the team a bit easier. Other points simply do not apply to the context, such as the need for each developer to release their modifications every day, since it is not a rule for developers to dedicate to the project every day. Given its expected workload of only fifteen hours per week, there are days when developers do not make changes to the software, so this principle could not be applied.

The team already to maintain a main repository of source code and accessible to all, counting on the versioning system Git, but it was not used in the best way. Commits and Merges were performed only when finalizing a functionality, that is, shortly before sending the modifications to the repository, in addition to using, each developer, a single Branch. From this analysis a new concept of temporary Branches was proposed and employed by the team. These Branches are now related to each activity (Issue) generated in GitLab[12]. Generally, each developer works on just one activity which maintains about eight active Branches. This is possible due to deletion of the respective Branches as soon as the activity is validated. There is a Branch that is seen as a central point for everyone to throw their changes by integrating with the main application code, called "develop_current". In this way, this Branch will be monitored by the Continuous Integration server.

Aiming to make easier the debugging of the code and tracking the modifications, developers were also instructed to perform Commits on every small set of changes in a class or method of software, besides as throw and seek modifications whenever possible, since the search for updates in the main Branch "develop_current" or the realization of Commits it was little applied, much for the forgetfulness of the developers. It was also

[9] `http://cruisecontrol.sourceforge.net/`
[10] `https://travis-ci.org/`
[11] `https://www.jetbrains.com/teamcity/`
[12] Open source software to collaborate on code - `https://about.gitlab.com/`

introduced the concept of Merge Request, in which the developer with intention to throw modifications in the Branch "develop_current" opens a request of Merge in the GitLab and, in this way, before the Merge is throw, the other developers can evaluate these changes, what before was an informal process, communicating to the leader the intention of Merge or even being postponed until the end of the week.

The Jenkins server was implanted to leverage Continuous Integration principles to be followed by the team. In view of this, the automation and execution of the build are performed on the server, a different environment from the local development environments, avoiding dependencies on local configurations. Even if the build was already automated from executing of the Gradle on Android Studio. The team has a suite of automated acceptance tests, but these were run only after the separation of a release by leader and running in the local production environment on the test analyst machine, the same used for the creation of these tests. Thus, the acceptance tests have been configured to run during the build process that is started from the modifications released in the Branch "develop_current" and provides, at the end, a feedback of the state of the system and the changes made.

At the end of the build process, logs are generated containing detailed of the process information, the compilation results, the tests, the static analysis, and more. In this log, in case of a build break, that is, a process interruption in the event of a version that did not succeed in some test be built, it is possible to find the problems, errors or nonconformities after the change.

In addition to logging, a dashboard presents feedback from these results with the help of the Build Monitor Plugin. The team should then be attentive to these results, which also present information from the member of the team responsible for the change and when it occurred. This developer should then arrange to fix the problem as soon as possible. When the problem persists, the Branch "develop_current" must be reverted back to a stable previous state and test the modifications of other developers, thus preventing the project from stagnating. The responsibility for the build, therefore, must be for each developer, who cannot validate his activity until he gets feedback from the introduction of his changes made available by Jenkins.

4. Results

The introduction of the Continuous Integration in the team had its start in sprint twelve, considering the accumulated sprints of the project, each with an average duration of four weeks. It was so presented with the proposal of improvements in the process and methodology of Branches used by the team, in which the first changes occurred. The Jenkins Continuous Integration server had its activities started at the beginning of sprint thirteen. The data and opinions collected correspond to the period from its introduction to the end of the sprint fourteen, characterizing the elapse of three sprints, about twelve weeks, presenting promising results presented next.

In addition to the data collected from the record history, generated frequently during each sprint and the project data available in GitLab, a two-step questionnaire was applied to the developers who are part of the team. The first step was applied shortly before the start of the practice[13] introduction and the second in the middle of the thirteenth

[13]Link to first questionnaire applied - `https://goo.gl/forms/A6ar5mTMU17denn02`

sprint[14]. The objective of this questionnaire was to collect the opinions and the perception of the developers about the first impacts of the implementation of Continuous Integration in the work culture. The questionnaire was elaborated considering the possible pertinent and observable points by the interaction of the team with the process of management and configuration of software employed, broaching issues related also to the own evaluation of the developers regarding the knowledge and compliance of the proposed work methodology.

Factors such as sprint happen in a recent period after the first delivery of the application and having a considerable number of reported bugs, influencing directly on the increased dedication of the developers to correct them, consequently extrapolating the expected of fifteen hours, directly impacting on the data collected. Another important factor is the thirteenth sprint happen in the university's end-semester period, which implies in reducing the productivity and dedication of some developers on the project, presenting as justification the great load of exams and works academics in this period. In sprint fourteen, the project suffered a reduction in the pace of development in the first weeks, a consequence of the return of the school recess where many students return from their hometowns and must to readjust to the routine.

With the adoption of the proposed improvements, it was observed that the amount of Merges showed a significant initial evolution in sprint twelve, however, given the factors discussed previously, in sprint thirteen there was a decrease in the number of Merges, however, it was little below sprint ten which recorded the highest values prior to adoption, seen in Figure 1. The sprint fourteen also showed an evolution considering the previous historical to adoption beyond the weight of the related factors.

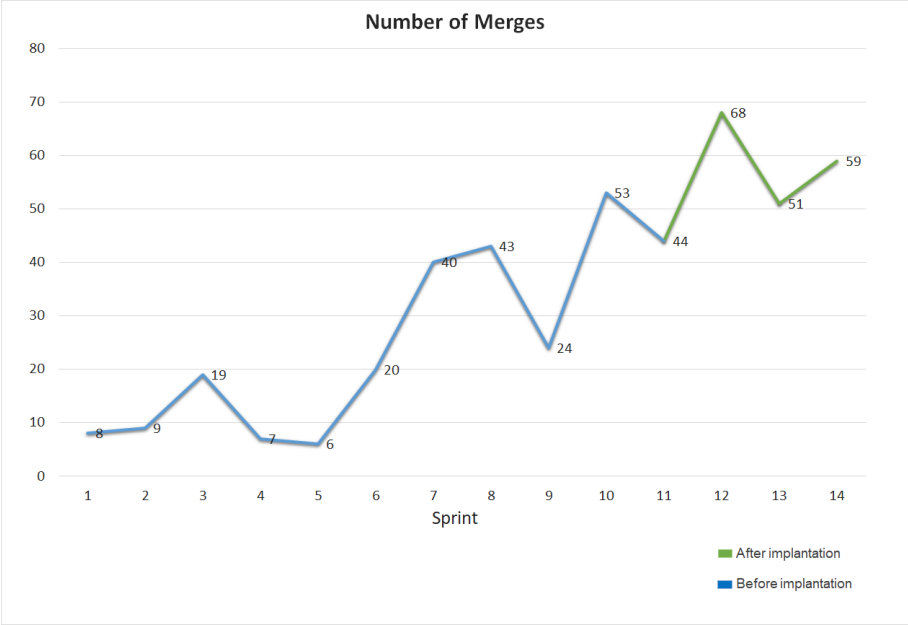

Figura 1. Amount Merges

The results of this evolution were similarly perceived by the developers, who responded to suffer less to problems of conflict of Merge, besides the reduction of the time

[14]Link to second questionnaire applied - `https://goo.gl/forms/7ShDwVpZsEAu72GL2`

to solve them by reviewing the modifications that introduced these problems, as seen in the relation of Figure 2. They also look for more updates on Branch “develop_current”, at least, always before submitting a Merge request to the “develop_current”. This factor may also be related to the small improvement in the habit of following the development methodology, revealed by the opinion of the developers themselves when argued about in the questionnaire.

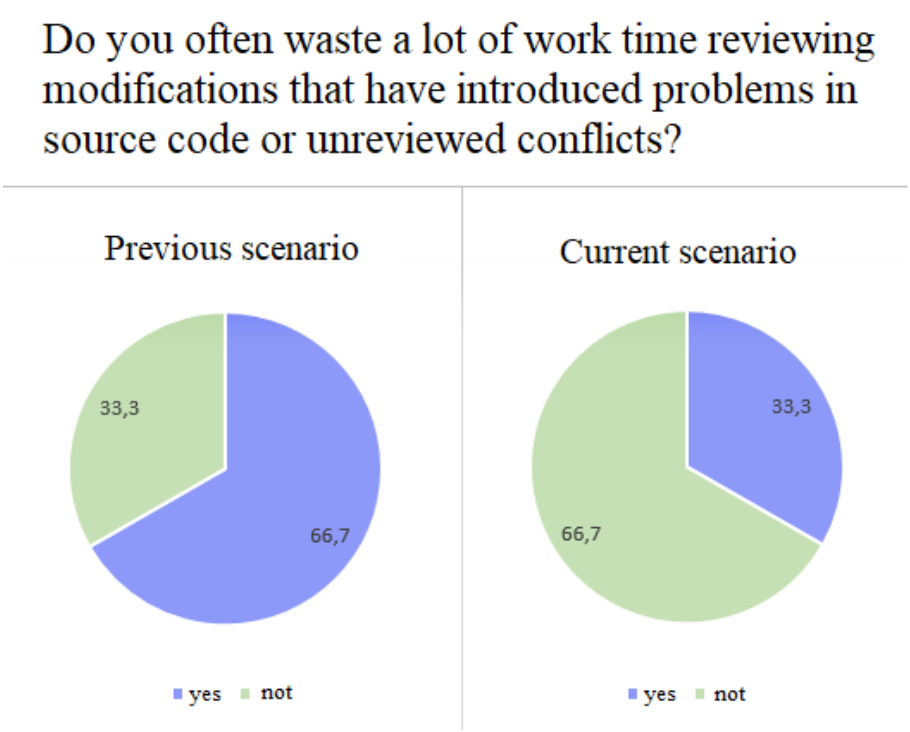

Figura 2. Research question

The points highlights by the developers were automation and branching-related functionality, making them more modular and making it easier to detect and debug bugs, which may arise in previously stable parts of the software. Argued about the future potential of the adoption of the practice by the team, four of the six assessed that this can be of great importance during the development also showing great level of enthusiasm for the continuation and improvement of practice. With respect to the improvements that could be leveraged towards the Continuous Integration, the developers cited the coverage and new types of tests, the generation of reports of more organized automated tests and the own maturation of the team in relation to the adopted principles adopted.

After the implementation of the Jenkins server, were carried some builds out with the integration of code in Branch “develop_current” and also in a manual way. The average duration of a completed build was fifty minutes, running only four to six builds during the week. About nine minutes of the process are dedicated to the static analysis performed by SonarQube, which can be improved with a more powerful server. Approximately thirty-five minutes are devoted to acceptance testing, the rest is Gradle build-related.

The first build did not count on the presence of the acceptance tests, occurring with success. With the introduction of testing in the process, what we got was succession

of broken builds due to the instability of some test scenarios plus required artifacts and files that were only available in the test analyst's development environment. Seven build breaks resulted from bug detection, six of them through the acceptence tests and one in the Gradle build process, these were quickly solved with low cost of dedication. Other breaks, for the most part, were due to network problems or with the tablet connection problems, hardware used to run the application together with the tests on the Continuous Integration server. After some builds was obtained a stabilityin the builds, characterizing a sequence with success, however, as a consequence of reasons cited above, later, others builds also failed, alternating this way between successful and failed builds.

5. Final Considerations

This article presented the implementation of Continuous Integration in an academic research project, with a satisfactory initial result observed by the development team responses through the questionnaire accomplished and considering the evolution of the number of Merges observed, showing positive impacts on short term, although since the maturation of the idea the implanting Continuous Integration in the project, numerous challenges arose during the course.

The team's median disciplinary level in following the process regularly, self-rated by the developers themselves in the questionnaire, was a major challenge, especially when proposing modifications to a routine that was comfortable and familiar to most developers. The lack of the developers experience in projects of this magnitude may be a reflection of the difficulty in properly following such methodologies. This change in the culture of the process of integration and version control has been happening gradually, the animation of the team towards the Continuous Integration is a factor that favors the continuity of this evolution. Initially, the development leader is still evaluating and being the only one with the autonomy to accept integration requests (Merge Request) with Branch "develop_current", but with the maturation of the use of the Continuous Integration by the team, approval of the request tends to from the developer itself who did it, according to the evaluation of the others.

The configuration of the integration environment presented setbacks in some points. Using an Android Virtual Device (AVD) was the first choice to run the application and its acceptance tests during the build. AVD has proven to be the easiest and fastest solution since Jenkins provides a plugin that helps you fully utilize it. But problems with managing Android Hardware Acceleration technology on Linux and running the Jenkins plugin when communicating between a virtual device emulated with ADB took considerable time to resolve. Finally, the performance limitations of the server, not getting a good performance when running the virtual device and running the application with the tests, made this option unfeasible. As a result, the output was to use one of the physical devices available to the developers (tablets), which favors the principle of running the application in their target environment, but makes the use of the device limited only for this purpose.

The long duration of the build was characterized as a negative point and a challenge, in addition to contradict to principle cited by [Fowler 2006] in having fast builds, which can generate a scenario of queued builds waiting to be executed by the Jenkins causing delays in feedback when launching some change in the repository. However,

the duration of the build does not cause major collateral effects in this project, given the project's characteristics such as the team size and dedication expected from the developers. In a future context with the addition of test scenarios together of software evolution, the adoption of night builds [Fowler 2006] can be considered to perform complete tests throughout the software, leaving only a select set of tests , usually related to essential functionalities in software, performed at the time of integration.

With the implementation of the Continuous Integration practice, the next step will be to improve the process in order to achieve a high level of team maturity for the implementation of Continuous Delivery, in order to reduce the time gaps between of releases [Humble and Farley 2014], guaranteeing, in an ideal scenario, which releases can be reliably delivered at any time.

References

Duvall, P. M., Matyas, S., and Glover, A. (2007). *Continuous integration: improving software quality and reducing risk*. Addison-Wesley Professional, Upper Saddle River, NJ.

Fowler, M. (2006). Continuous integration.

Hembrink, J. and Stenberg, P. (2013). Continuous integration with jenkins. *Coaching of Programming Teams (EDA 270), Faculty of Engineering, Lund University, LTH*.

Hukkanen, L. (2015). Adopting continuous integration - a case study. Master's thesis, Aalto University, School of Science, Espoo.

Humble, J. and Farley, D. (2014). Entrega contínua: Como entregar software.

Maple, S. and Shelajev, O. (2016). Java tools and technologies landscape report 2016.

Pereira, I. M., de Senna Carneiro, T. G., and Pereira, R. R. (2013). Developing innovative software in brazilian public universities: tailoring agile processes to the reality of research and development laboratories. *Proceedings of the 4th Annual Conference on Software Engineering and Applications (SEA 2013)*.

Polkhovskiy, D. (2016). Comparison between continuous integration tools. Master's thesis, Tampere University of Technology, Tampere.

Rezende, D. A. (2005). *Engenharia de software e sistemas de informação*. Brasport, 3 edition.

Rodrigues, N. N. and Estrela, N. V. (2012). Simple way: Ensino e aprendizagem de engenharia de software aplicada através de ambiente e projetos reais. *Anais do VIII Simpósio Brasileiro de Sistemas de Informação*.

Ståhl, D. and Bosch, J. (2014a). Continuous integration flows. In *Continuous software engineering*, pages 107–115. Springer, Switzerland.

Ståhl, D. and Bosch, J. (2014b). Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, 87:48–59.

Stolberg, S. (2009). Enabling agile testing through continuous integration. In *Agile Conference, 2009. AGILE'09*, pages 369–374, Chicago, IL, USA. IEEE Computer Society.