TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

DENIS POLKHOVSKIY

COMPARISON BETWEEN CONTINUOUS INTEGRATION TOOLS

Master of Science thesis

Examiner: Dr. Terhi Kilamo
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 6th April 2016

ABSTRACT

DENIS POLKHOVSKIY: Comparison between Continuous Integration tools
Tampere University of Technology
Master of Science thesis, 54 pages, 0 Appendix pages
May 2016
Master's Degree Programme in Information Technology
Major: Pervasive Systems
Examiner: Dr. Terhi Kilamo
Keywords: integration, development models, tools, continuous integration

Nowadays many tasks are performed by the help of various software in each aspect of our life. This software involvement becomes deeper and deeper every day, which leads to a need to release products faster to the market, considering the intense competition among companies. For this reason, companies start to enhance benefits of development models with use of Continuous Integration. Today, there are many CI tools available on the market that can be used for the software integration process and choosing right tool is becoming a hard task.

During this research, it was clarified and shown the overview of most common CI instruments. Then, it was performed comparison and was provided decision matrix for Continuous Integration frameworks. Based on comparison, Jenkins was preferred over other tools, since it is a free open-source project with providing great flexibility to many different development methodologies. This Continuous Integration framework fits to majority defined requirements and meets our thesis research questions. In addition, this thesis project can be helpful as short guide, where is suited basic knowledge for starting work with CI and it allows to dive into CI process, and learn in parallel from where it takes origins.

PREFACE

This Master of Science Thesis was written in the Department of Pervasive Computing at Tampere University of Technology.

I have found the writing of this thesis to be the hardest work within my period of study in Finland. I am sure it would not be possible to complete the thesis without the support of many people.

I would firstly thank my supervisor Dr. Terhi Kilamo for her guidance, sharp technical eye, patience, comments, advice and suggestions for research directions in writing this Master's Thesis. As well, I would like to primarily express my gratitude to Professor Mikko Tiusanen for his contribution and provided opportunity to start writing the thesis with him. I would like to thank the Tampere University of Technology for the opportunity to receive a Master's Degree. I would like to thank all my close people, especially my brother who has always inspired me to work. Finally, I would like to thank my mother for her constant encouragement.

Tampere, 22.05.2016

Denis Polkhovskiy

TABLE OF CONTENTS

1. INTRODUCTION

Nowadays many tasks are performed by the help of various software in each aspect of our life. This software involvement becomes deeper and deeper every day [35], which leads to a need to release products faster to the market, considering the intense competition among companies. For this reason, companies cannot create software products without using software development models [50]. Today, it goes progression from traditional software development methods towards Agile methodology in the software business [62]. In Agile methods you get feedback from customer frequently, by these actions you increase customer involvement and improve a quality of software development, allowing to meet customer expectations [45]. Moreover, today companies start to enhance benefits of development models with use of Continuous Integration. Generally, it allows to release and repair any issues faster, and get quicker feedback. According to Martin Fowler [46], by using this technique you can save a lot of money and time during integration phase, he claims that integration phase is usually most unpredictable and time-consuming part of development. In software development models, software integration is the process of bringing software parts together into one working system. Integration phase is performed at the end of a development workflow. Because of that, a lot of issues and problems occur in the code that contain interaction between individual parts of the system, as a result it extends time needed for correction of this problems and it makes more complicated to further predict amount of possible errors [45]. Consequently, it brings a significant increase of overall project cost and release time [46]. When the software becomes bigger in size and complexity of project is increased, it is getting hard for developers to maintain high level of quality. There is no easy solution to this problem, but one possible way of solving integration problem and tracking quality of the project is start to using Continuous Integration practice in the software development [46]. Continuous Integration is one of the existed methods in software business, which is used for improvement and automation of the software development. In comparison to traditional software integration phase, where this phase is a large step of development, integration occurs constantly [45]. The main advantage is that software is

integrated after any changes in subcomponents or subsystems of a system [46]. As a rule, each new change submitted by a developer is being built using a code base and automatically checked by predetermined techniques like automated testing, static code analysis and etc. The Continuous Integration tools are automated and there is no need for a developer to interfere with CI process [45].

Nowadays, there are many CI tools available on the market that can be used for the software integration process. Clearly, with so many existing Continuous Integration tools, it might be challenging to choose which one fits your needs. Today, most of the companies at the market provide CI products with similar services [46]. Moreover, it does not mean that the most expensive commercial product is better than low cost or open-source tool [45]. Generally, all of these tools are trying to make the process of building software automated. In spite of this fact, Continuous Integration tools have their own strengths and weaknesses, therefore choosing right tool for a job is really important. The wrong choice can reduce overall flexibility; it can increase time for performing ordinary actions or it can lead to some others problems in Continuous Integration process [46]. We just need to investigate them, and then summarize features they have and what kind of work they can do before taking decision. However, this field is a quite new and grows very fast, new tools are released almost every year and many developers are involved in this field. It makes this area very interesting in terms of availability of products that are presented on the market today.

The nature of the thesis is a look into Continuous Integration as a part of the software development process. Whereas, the main problem in this thesis is exploring CI tools and making a comparison of them with each other by defined criteria and answer to question which tool meets to our requirements.

The object of this thesis consists of software methodology study where Continuous Integration is applied. Moreover, studying Continuous Integration process cannot take place without investigation of the software development process field. Then, it includes survey of literature and web-based sources, in order to find criteria and requirements for examining CI servers.

Finally, the main goal of the thesis aims to show analysis and comparison of selected tools and to choose tool for CI implementation which meets ours criteria. Thus, the research questions that we want to state in this thesis include:

-"Which criteria and requirements are appropriate for examining CI tools?"

-"Which tool is good as entry-level tool for getting acquainted with CI area?"

-"Which tool satisfies mostly to our criteria and requirements?"

According to end results, we have studied thoroughly CI process. Then, it was clarified most common CI instruments and shown the overview of CI tools. In the end of thesis, it was provided comparison table. Next, we have answered for research questions and we have chosen one CI tool, which meets best for defined criteria.

This thesis consists of 7 chapters, where chapter 2 presents preliminary knowledge about a modern software development models. Next, in chapter 3, we provide essential information about CI as a single process and explain all relevant details. Further, in chapter 4, we state out thesis approach and explain it in a detailed manner. Then we provide results of literature and web-based review in chapter 5, describing important features and defined main criteria and requirements for CI tools comparison. Also, we present our surveys related to existing CI tools and its features. Finally, we conclude this chapter with providing a table with comparison of CI tools based on predefined data and the evaluation of them. Further, in chapter 6 of this thesis we show our observations and give answers to research questions. Lastly, in final chapter we argues about the strengths and weaknesses of the thesis and we set out possible future research ideas.

2. DEVELOPMENT MODELS

The term "Continuous Integration" goes hand in hand with system development models [13]. The Continuous Integration emerged at same time with the evolution of these models. As a key to better understanding of the CI as a process, we require a comprehension of main principles of development techniques.

Software Development process can be described as a sequence of steps, which a company does regularly during software product development [38]. Different models of this process exist, but it can be divided to traditional and lean models. Each of software development models describe their approach in the form of tasks or operations that take place during the process [50]. Nevertheless, it is important to select the right software development model, because this choice has high influence on a success of the development [58]. However, thoughtless choice can cause a rise in software costs [38]. In addition, it can cause the violation of deadlines and quality decrease of developed software [45].

Software development methodologies have evolved since 1970's [36]. The first model that appeared in this evolution was the waterfall model. Then it arised such models as prototyping model, incremental model, spiral model, RAD model. Finally, Agile saw the light in 2001. However, it was formulated and introduced in 1974 by Edmonds in his research paper [65]. Presently the process of software development is going through an evolutionary movement, from traditional software methodologies like waterfall model in the direction of agile methodologies. Many companies are moving to agile software development to improve quality and reduce delivery time [36]. This methodology serves as excellent choice to be adopted in modern software development process to replace the traditional heavy weight development life cycle [51]. According to BengLeau et al. [51], they differentiated traditional approaches as heavy weight methodologies and agile methods are called as light weight method. In this paper [51], approaches are divided on the following basis - user requirements, rework cost, development direction, testing, customer involvement, extra quality

and project scale.

Helena Olsson and Jan Bosch presented a developed model "The stairway to heaven" in their study [56], which shows how software companies follow when evolving their software development processes. "The stairway to heaven" model is presented in Figure 2.1 below. We decided to include their "new" model, because it describes and shows the evolution path from traditional development to agile practices and further to continuous integration and beyond. They used this model as a background for understanding the challenges associated with moving beyond agile practices and towards continuous integration and deployment of software.

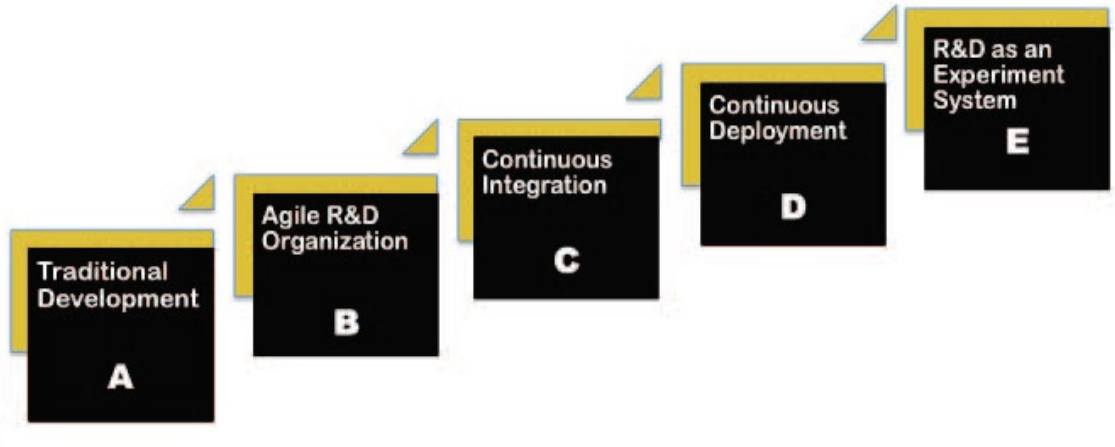

Figure 2.1 *"The stairway to heaven" [56]*

"The stairway to heaven" has several stages and for each of them, there are typical barriers and key focus areas that need to be handled to move from one step to the next. Whereas, first stage is transition from traditional development to agile. Then, it follows achieving continuous integration and adopting continuous deployment. Consequently, the final stage of "The stairway to heaven" is where the entire research and development system reacts and works based on instant customer feedback and actual deployment of software functionality is seen as a way of experimenting and testing what the customer needs. At this stage, deployment of software is seen more as a starting point for further "tuning" of functionality rather than delivery of the final product [55].

Generally, companies choose development models based on their own requirements, it often occurs to be a reason that some methodology fits better the development of a particular software. However, Agile is the most popular methodology nowadays

[3]. Nevertheless, another common methodology is a "Waterfall Model", but in comparison with Agile, it is quite old model and it is not so flexible [38]. In contrast to Agile, waterfall model detects most of issues only at the end of procedure, because testing is performed after development stage [33]. Whereas in Agile tests occur much more often, because development is divided into "sprints" and working this way leads to early detection of many problems [5]. In addition, it saves a lot of company's money in the long term perspective [50].

2.1 Traditional Model

Waterfall model the best known and oldest representative of Traditional Software Models [38]. Dr. Winston W Royce published the article "Managing the development of large software systems" in 1970 [57]. Today, this article is considered as the first work that describes the waterfall model. Royce described waterfall model as more grandiose approach of software development compared to the small principle of analysis and coding, which was widely distributed at this time [57]. It was the first model, which formalized the structure of the stages in software development, with

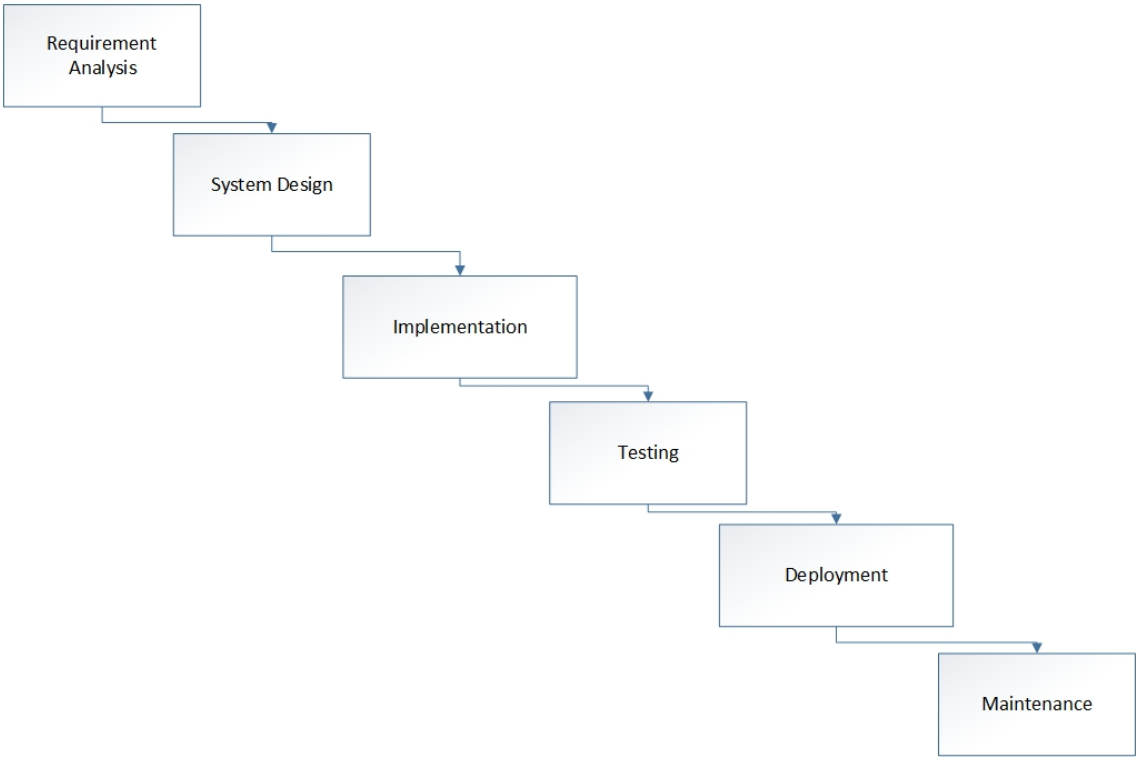

Figure 2.2 *General overview of Waterfall model.*

emphasis to the initial requirements, design and intention to create documentation at an early stage of the development process [18].

A distinctive feature of the waterfall methodology is that it is a "top-down" development technique with independent phases, the process is continued by using the ordered series of steps, because of that it produces a framework for software development. Commonly, unmodified waterfall model includes six distinct phases, which is shown in Figure 2.2.

In waterfall model, each next stage of software development starts only after full completion of the previous stage [33]. Hence, the output from completed stage is used as an input in the next process. For instance, the output of software requirements phase is software requirements' document that is applied as the input document to analysis stage. Each phase has specific requirements for input and output.

The transition from one phase to another goes through a formal review. This means that customer gets an overview of the development process. Besides that, the software quality is checked. As a rule, passed review phase means that agreement between the development team and the customer has been achieved. When this phase is accomplished, developers can move to the next phase.

As concerning advantages of waterfall model [18]. It has an easy entry period for new programmers into project during the maintenance phase, because of serious technical documentation. In Waterfall, phases are quite well defined and accessibly. There is no overlapping of phases, only one phase at a time during development period [32]. The time of completion of each phase is used as a step and due to milestones clearly defined by developers [26]. It allows members of the project, who completed their work on the carried out stage, to take part in other projects. Then, this approach fits in small projects very well. Moreover, reduction of rollback probability, since the process of development is consequent [26]. Furthermore, since requirement and design phases receive a lot of attention and are accomplished before implementation phase, minimum loss of time and effort is achieved, as well as reduction of a chance to fail customer expectation [26]s. In addition, this model is very easy to understand, since main goal of a process is to perform all necessary actions.

In reference to **disadvantages** of Waterfall model. Firstly, it is not good idea to use Waterfall in object-oriented projects [26]. In large projects, process of evaluation of

the required time and budget for a each stage is very complicated. Next, Waterfall is not flexible as Agile methodology [4]. Also, there is no preproduction model being made before the finishing of the development process. Testing is performed after the completion of development; it means that there is a great chance to find more bugs and going to previous phases for fixing them is very difficult. Whereas, bugs have characteristic to be cumulative, with increasing number of bugs, it is getting harder to fix every single one [46]. The Waterfall model is not designed for dynamic changes in the requirements of a project during the life cycle, all requirements must be defined in the beginning of development, but customers often do not know exactly all needed requirements, if they keep adding requirements after the requirement phase, it makes model less effective and causes many problems. As a result, re-engineering of the project may decimate its budget. It may become hard to define the amount of work done. A statement, such as "35 percent of work is done", is pointless and it is not an indicator of a current state of a project for a project manager. The customer hardly has the opportunity to become familiar with the system in advance, it happens only at the end of the life cycle. The customer is not able to see intermediate results, and user feedback cannot be sent back to the developers. Since a finished product is not available until the end of the process, the user takes part in the development process only at the beginning (during gathering of requirements) and at the end (during deployment and maintenance phase). Users cannot ensure the quality of the developed product before the end of development process. They are unable to evaluate the quality because they do not see prototypes of finished products. The user has not a possibility to gradually get used to the system. The learning process occurs in the end of the life cycle, when the software is already deployed. There is a need for strict control and supervision, because the Waterfall model does not provide ability to modify the requirements iteratively.

Beside of waterfall's disadvantages, this model remains one of the most popular methods in the IT area [18]. Waterfall model's functions very good in projects, where requirements are well defined and understandable in an early stage, and cannot be changed during development. In addition, it is recommended for projects with small budget [33].

2.2 Lean Software Development Models

Lean and agile are very close terms with very similar philosophy [5] and they share many of the same principles and many Agile principles are borrowed from Lean [23],

although lean is more general term and lean's ideas fit in very well with the agile software development. Also, according to Fowler article [5], he claims that if use agile you also use Lean at same time or vice versa. Agile software development is a set of principles and rules which determine what software is developed [42]. Basic principles of Agile is shown on Figure 2.3. Agile methodology is not a single method of development, but a group of software development methods. The term of Agile has emerged during a meeting of seventeen software developers, who named themselves as "The Agile Alliance". After that, it was published as the "Manifesto for Agile Software Development" in 2001, which was signed by all participants of this meeting. The idea of this manifesto has emerged based on need of alternative model for heavyweight software development models, like waterfall approach, which was standard at that time.

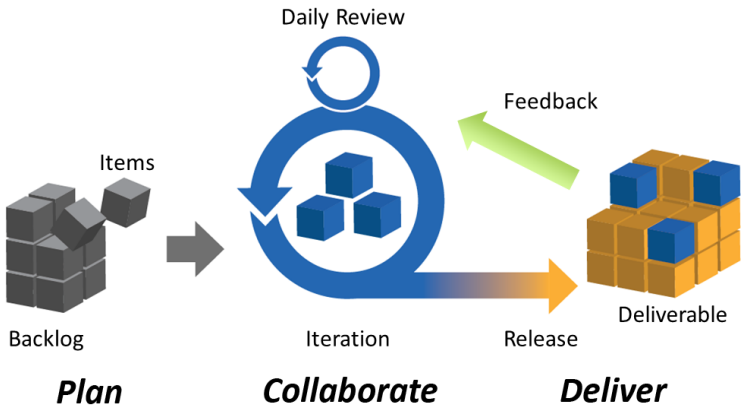

Figure 2.3 *Typical view of Agile model [44].*

Most agile methodologies aim to minimize risk by developing software, as a series of short cycles, that are called iterations, they typically last from one to two weeks. Each iteration itself looks like a miniature software project and includes all the tasks required for software functionality growth: planning, requirements analysis, design, coding, testing and documentation [2]. Although, a single iteration is usually not enough for releasing new version of the product, it means that the software project is ready for release at the end of each iteration. After each iteration, the team performs a reassessment of development requirements [60].

Agile-methods focus on direct face-to-face communication [58]. Most agile-teams

placed in the same office; sometimes it is called the open-plan (bullpen) office. In addition, it includes a product owner. It can be a customer or his/her authorized representative, who defines product requirements. This role can be performed by a project manager, business-analyst or a client. Furthermore, the office may also include testers, interface designers, technical authors and managers.

Following Figure 2.4 presents main **values**, described in Agile manifesto, for software development process, where authors evaluate values from the left part more than from the right [42].

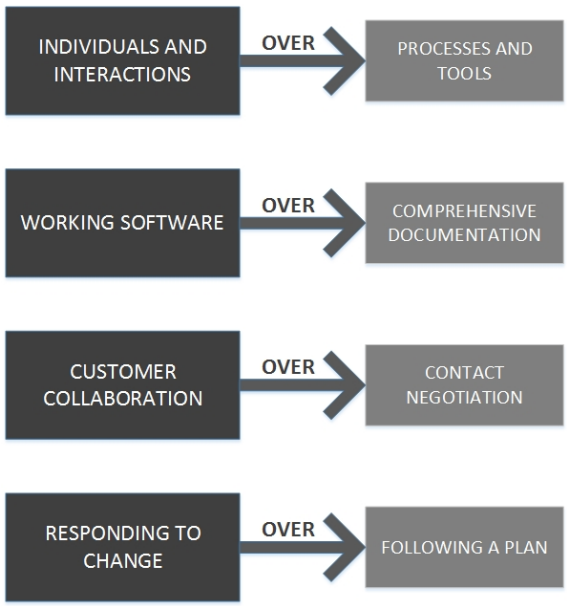

Figure 2.4 *Values of Agile manifesto.*

Individuals and interactions over processes and tools. It means that communication and collaboration between developers is the main factor that leads to success.

Working software over comprehensive documentation. Creating working software is main aim of development process rather providing perfect documentation

Customer collaboration over contract negotiation. Involving customer to process is very important, only by working closely and frequently with him; software achieved good quality and met customer expectations. Because of good customer collaboration, agile projects are more than two times successful in comparison with traditional projects.

Responding to change over following a plan. Customer do know all requirements in the beginning of development. It is common problem that customer wants to add new requirements or some features during a development process to a project. Whereas Agile techniques allows it, in comparison with traditional models where all steps are planned and customer can be unhappy in the end, because he got not exact product what he wants to see.

Beside of stated values in Agile manifesto, it was defined **12 principles** how agile process have to follow [42]. These principles are in original order:

"- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity - the art of maximizing the amount of work not done - is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

2.2.1 Scrum

Scrum is an incremental and iterative Agile method [58]. Scrum clearly focuses on the quality control of the development process. Scrum is a set of principles used in development process, that allow, in strictly defined periods, provide working software to the customer with incremented features. Required functionality is determined at the planning meeting before the start of the next sprint and cannot

change during the whole sprint. Small fixed sprints add predictability and flexibility to the development process. This approach was described first by Hirotaka Takeuchi and Ikudziro Nonaka in 1986 [63]. They noted that the projects with small, cross-functional teams, usually systematically produce better results.

Following team structure for each sprint is allocated by the Scrum method [58]:

- Scrum Master.
- Product owner.
- Team.

Scrum Master has the most important role in the methodology. Scrum Master is responsible for the success of Scrum in the project. In essence, Scrum master is the interface between the management and the team. The basic role of the Scrum Master is to maximize efficiency of the team by removing all obstacles and outside influences, also he/she provides assistance, training and he/she motivates the team. All these actions allow team to become more focused on the work. As a rule, this role in the project is taken by the project manager.

Scrum Master includes following main responsibilities:

- Creating an atmosphere of trust.
- Removal of obstacles.
- Making problems visible and questions obvious.
- Responsible for compliance practices and processes in the team.

Scrum Master organizes daily 15 minute meetings and he/she reviews the progress with the team [58]. He/she follows progress through the list of tasks that must be completed during a Scrum sprint, which called Sprint Backlog. Also, he/she notes the status of all tasks in the sprint. Scrum master can also help product owner with creation of Sprint Backlog for the team.

Product Owner is a person who is responsible for development of the project. Product Owner makes the final decision on the project, because of that it is always

one person, not a group or a committee. He/she is the link between the development team and stakeholders. He/she pursues a goal to maximize the quality of developed product and work of the team. One of the main tools of the product owner is the product backlog. The product backlog contains tasks that are required to accomplished. They are sorted in priority order. Product owner responsibilities include:

- Forming a vision of the product.
- Managing profitability.
- Managing expectations of customers or all stakeholders.
- Coordinating and prioritizing tasks in the spring backlog.
- Providing requirements to the team in the clear form.
- Interacting with the team and stakeholders.
- Accepting code at the end of each sprint.

Product owner assigns tasks to the team, but he/she is not entitled to assign tasks to specific members of project team during the sprint.

Developing Team in Agile methodology is self-organized and self-managed. The team is liable to Product owner for implementation of designated work in the sprint. Teamwork is evaluated as the work of single group. In Scrum, the contribution of individual members of the project team is not measured, because it disarranges self-organized teams [25].

According to The Scrum Guide, developer team has following duties [64]:

- Responsible for evaluating elements of sprint backlog.
- Make decisions about design and implementation,
- Be self-organizing, nobody can specify to the team how to implement tasks of backlog.
- Whole team is responsible for the work - not individual members.

- Be multifunctional: team should have all necessary skills for a product release.
- Developing software and providing it to a customer.
- Have control over their own progress (together with the Scrum Master).
- Liable to Product Owner for the result.

Size of the team is limited. Typically, it is around seven persons. Scrum team is cross-functional, it consists of people with different skills like developers, analysts, testers. There are no predefined and divided roles in the team, that limit the scope of team members. For purpose of easy communication, team must be located in the same place. The team should be provided all necessary things for a comfortable work, such as blackboards, flip charts, essential tools and development environment. Result of the sprint depends on all of these conditions.

Figure 2.5 describes schematic representation of **Scrum workflow**. The basis of pipeline is a sprint during which whole work on the product is performed. At the end of Sprint new working version of the product is obtained. Sprint is always limited in the time (1-4 weeks).

Sprint Planning is organized before each Sprint, where Product Backlog is estimated and eventually Sprint Backlog is formed. It includes tasks (Story, Bugs, Tasks), that earlier have to be solved in the current sprint. Each sprint should have a purpose. It serves as motivating factor for the team and this purpose is achieved by accomplishing tasks from the Sprint Backlog [64].

Another important part of scrum methodology is a Daily Scrum, where each team member has to answer following questions: "What I did yesterday?", "What I am planning to do today?", "What kind of obstacles I have encountered?".

The purpose of Daily Scrum is to determine status, progress of work in the Sprint. Also to early detect possible issues and develop solutions that will allow to adapt the project strategy in order to achieve Sprint's success.

At the end of Sprint, Sprint Review and Sprint Retrospective take place. The purpose of them is evaluation of effectiveness (performance) of the team in the past Sprint. Also, other tasks are prediction of expected performance (productivity) in the next sprint, identification of existing problems, estimation of the probability of completion all the necessary work on the product and etc.

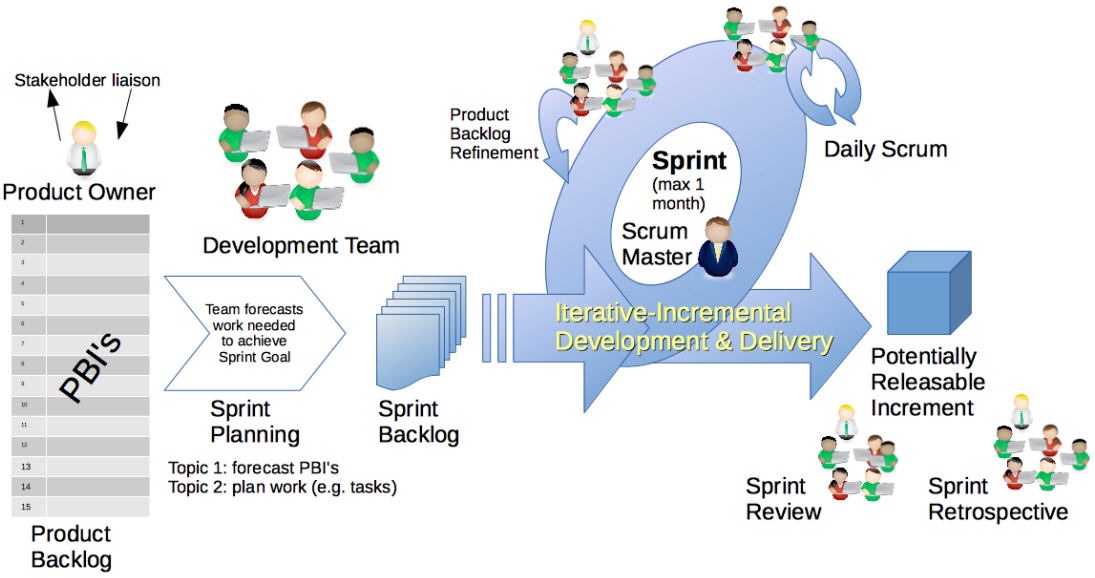

Figure 2.5 *Schematic representation of Scrum workflow [44].*

2.2.2 Extreme Programming

According to Lowell Lindstrom [52], Extreme Programming is most pervasive Agile Software Development Methodology. He characterized Extreme Programming as exciting, popular, controversial approach of software development. Based on this information, we decide to introduce XP as another important representative of Agile methodology.

Extreme Programming is incremental and iterative Agile method. It is lightweight, efficient, flexible, low-risk and predictable way of software development.

Creator of XP methodology is Kent Beck. In 1996, this technique was used first time during a failed attempt to save a project that developed a salaries calculating system in Chrysler. In 2000, this project was closed, whereas XP had already gained popularity and spread among software developers.

XP inherits all general principles of agile methodologies, which are reached with the help of twelve engineering practices. Most interesting technologies and practices of XP are provided further [52].

The project team must include **representative of the customer**. He/she has

detailed knowledge about required functionality and he/she prioritizes specific requirements. Then, his work involve evaluation of developed project quality. Technically, the customer representative can be an employee of Development Company. For instance, it can be a product manager or a business analyst.

The life cycle of XP project consists of series of releases [52]. Each release is a complete version of the product, which customer can use. In addition, each release contains additional functionality compared to the previous release. Release occurs every one or more iterations, which last from one to four weeks [37].

In XP practice, it is not recommended to spend a lot of time on planning. The planning process is called the game. A detailed plan is prepared only for the next iteration and for the next one or two releases [52].

2.2.3 DevOps

We decided to include DevOps since continuous integration is a key component of DevOps [14]. Nearly all in DevOps is continuous: such as integration, deployment, delivery or testing [11]. Moreover, Devops is extension of Agile practices and was basically born from increasing popularity of agile development [34].

DevOps is an approach based on lean and agile principles in which stakeholders and the development, operations, other departments cooperate to deliver software in a continuous way that enables the business catch market opportunities more quiclty and reduce the time to include customer feedback [59].

The term DevOps is put together from the two terms "Development" and "Operation", which are two classic departments in software companies. Development is responsible for all changes, bug fixes and development of the software. Operations is responsible for the software's stability. Both departments have interest that often conflict between each others [49]. The area around DevOps is expected to solve this conflict and bring back agility to the static and slow current "Development - Operations" structure, which is continuously blocking itself [49]. Graphical representation of DevOps structure is shown on Figure 2.6

There exist many different definitions of DevOps, but Michael Huttermann in his book "DevOps for Developers?" [49] gives his explanation of this practice:

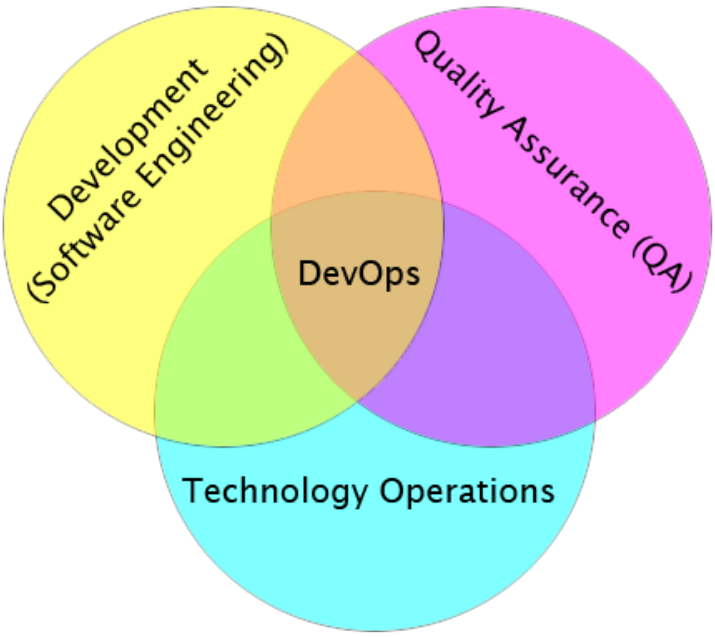

Figure 2.6 *DevOps as the intersection of development, technology operations and quality assurance [44].*

"DevOps is a mix of patterns intended to improve collaboration between development and operations. DevOps addresses shared goals and incentives as well as shared processes and tools. Because of the natural conflicts among different groups, shared goals and incentives may not always be achievable. However, they should at least be aligned with one another. DevOps respects the fact that companies and projects have specific cultures and that people are more important than processes, which, in turn, are more important than tools. DevOps accepts the inevitability of conflicts between development and operations [49]."

3. CONTINUOUS INTEGRATION

The concept of Continuous Integration started to commonly appear at the beginning of 2000's after publishing article [46] by Martin Fowler and Matthew Foemmel. However, this phrase has seen first use in "Object-Oriented Analysis and Design with Applications" [47] by Grady Booch in 1994. Afterwards, Extreme Programming was invented, including Continuous Integration term, by Kent Beck during his work on the Chrysler Comprehensive Compensation System payroll project [46]. Then article was published by Beck where he put stress on the importance of Continuous Integration in 1998 [40]. Furthermore, Beck accurately described Continuous Integration in his first full book on Extreme Programming [41]. Finally, Paul Duvall, Steve Matyas and Andrew Glover proceeded the work related to CI and they have announced a whole book about CI. This book is still continues to be the one of the main resources about the CI. The CI nowadays has become a popular technique for developing software. We can see popularity tendency based on Google trends service on Figure 3.1 has risen almost twice.

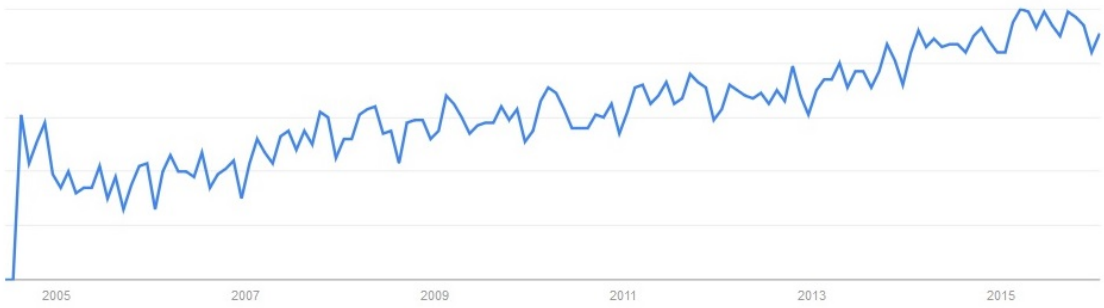

Figure 3.1 *Correlation of "Continuous Integration" search requests in a Google Trends*

One of the biggest problems in software development is integration [46]. In a small project with one developer it does not cause many problems, but if we imagine a software project in which several programmers are making code individually and after some time they need to merge and built their codes, this process might cause

a lot of errors, conflicts and quality problems. Integration part is unpredictable process, it can be very expensive and lead to project delay [45]. Continuous Integration deals with these risks much faster due to small daily increments in a project. Fowler and Foemmel in their popular article [46] give the definition of CI as follows:

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly [46]."

According to Duvall's book [45], this definition implies that:

- To be sure that any changes in project will not break the integration, developers have to build software on their own work machines before they commit code to the version control repository
- All developers have to submit their code to a revision control more than one time per day or at least once.
- One build machine is used as CI server, where integration happens several times a day.
- Every build have to pass one hundred percent of tests to be recognized as eligible.
- Correction of errors has the highest priority.
- The results of the build and unit tests are automatically posted to a web site, such as the Hudson web server, where all team members can be informed of the current state of the software.
- All developers of project can see current results of integration through a web site, which used for notifying team members by CI software, such as The Jenkins or The Cruise Control.
- CI reduces the collective responsibility of the group and reduces the complexity of the project, due to reduction of the amount of manual work performed at each change in the developed software.

Whereas, Patrick Cauldwell in his book "Code Leader: Using People, Tools, and Processes to Build Successful Software" [43] shortly describes the philosophy of CI as "integrate early and often", because more often you integrate the less work it is for everyone, and includes there two biggest goals of any CI effort:

1. To make sure that there is always a testable version of your software that reflects the latest code.
2. To alert developers to integration problems as quickly as possible.

3.1 Elements of Continuous Integration System

A general Continuous Integration pipeline starts from submitting code to the repository. Any team members of average project can trigger CI workflow, it can be developers, database analyst or some other team members, who change code or make any changes in a project. The scheme of classic CI server working cycle is illustrated in Figure 3.2.

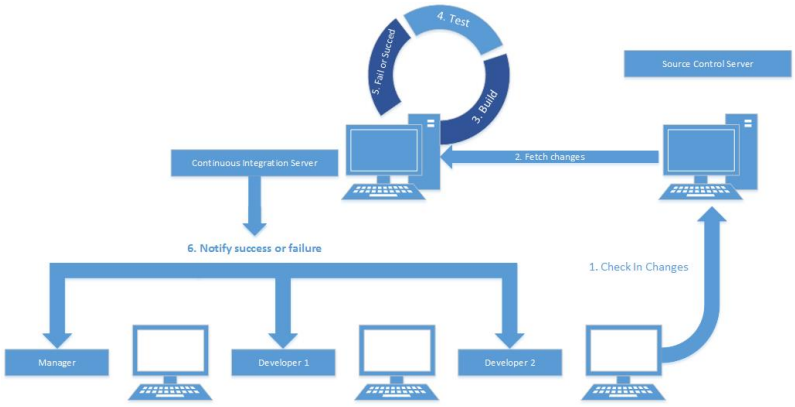

Figure 3.2 *The CI server working cycle.*

An average CI system [45] has following order of typical steps:

1. Developer submits code to the version control repository.
2. The CI is checking the version control repository for changes, with some interval.

3. When CI server detects any changes in the repository, CI server gets last version of code for the version control repository and then performs building. Script, responsible for integrating software, is created.

4. After that, CI server sends feedback with result of building to team members of this project. Feedback can be sent by different methods, such as SMS, e-mail, lava-lamps and etc.

5. Then, The CI server goes to step 2 and continues this behavior before any changes in the repository occur.

Following notations clarify elements of CI work-flow [45].

First rule of **developers** is to run private builds. Each developer has to run private builds after finishing his tasks. If integration of this code with the rest of the code of the team is successful, he/she submits his/her code to the version control repository. It is very important step, since it prevents from submission of a broken code.

Version Control Repository is essential part of CI system. It is also known as Software Configuration Management. Commonly, all software projects have version control repository, even if they do not use CI. The main purpose of such repository is to control changes in the source code and other software elements; for instance, in documentation. This repository is used as unified codebase for all source code. It stores all versions of source code and all files, that were included in the project during process of development. According to Duvall's book [45], it is recommended to read "Software Configuration Management Patterns" by Stephen Berczuk and Brad Appleton for learning effective techniques of source configuration management processes.

Continuous Integration Server is the most important part of the whole system. A Continuous Integration Server creates an integration build when any changes in the version control repository are made. Usually server is configured to track any changes in version control repository within period of few minutes. On the other hand, if you do not want to wait this time before CI detects a change in a repository, you can reduce the delay by setting up a post commit hook. Therefore Version Control Repository can notify CI server that a build needs to be created. CI server gets files of source code and runs build scripts.
Typically, CI servers give user convenient options for displaying builds results. The

simple form is a web page, some of CI server's software already comes with a web application. Hence you can see the results and build reports in this application. As a result, using a CI server allows to reduce the number of custom scripts that are needed to be written in any case. Nowadays, many CI servers are free and open-source.

Build Script can be single one or a set of scripts, which are used to compile, inspect and deploy software. Build script can be used even without implemented CI system. The most famous build script tools are called Ant, NAnt, MSbuild, Rake and Make. This tools make software build cycle automatic, thus it saves a lot of time for developers.

Feedback mechanism is considered as one of the most important tasks in CI, because it provides immediate results after creation of integration build. If the feedback shows that the build failed, developers can start fixing problem immediately. The feedback can be given to user in different ways. There are a lot of variants: it depends on creativity of developers. For example, the simple form is a SMS or e-mail, but in some cases it can be lava lamp or even traffic light. On Figure 3.3, lava lamps notifies developer about success and fail results of building, where green is success and red lamp is fail.

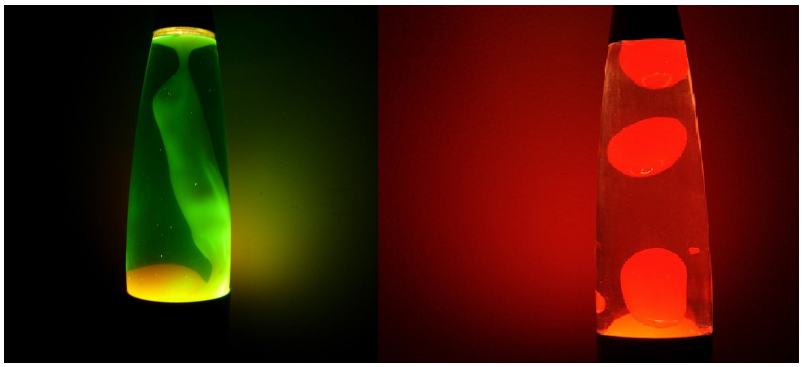

Figure 3.3 *Lava lamp shows results of building.*

Integration build machine is a machine which used only for one assigned task: be responsible for integrating software. The integration build machine contains the CI server, which tracks any changes in version control repository.

3.2 Integration Process

According to "Continuous Integration: improving software quality and reducing risk" by Duvall [45], there are four compulsory features used to implement CI. Firstly, it is a connection to a repository. Secondly, you need build script. Then, it is important to have some feedback mechanism. Lastly, you need CI server for integration of changes in a source code. These compulsory features are considered as the base behavior for a successful CI system. Components of integration process is explained below.

Source Code Compilation is one of the simplest and most common features in CI system [45]. Compilation is a process of creation of human-readable executable code from source code. Source code compilation depends on the type of the programming language in a project. Compilation process may vary if you use dynamic language in your software development. Examples of dynamic languages are Python, PHP and Ruby. When you use these languages, you do not create binary code, but perform strict checking, which can be considered as compilation in this case. Dynamic languages have some benefits in other actions during a CI build.

Database Integration is one of the most essential details of software application is the database. It must be involved in the integration phase. The implementation of CI system can guarantee integration of database by means of version control repository in a project. In this case, repository serves as unified source.
Continuous Integration in databases includes three main principles of operation:

- It verifies the state of the database after changes were made.
- It always runs tests after any changes have been applied.
- It updates when any changes in a target database are made.

Combination of these techniques shows that Continuous Integration can detect any problems which have not been noticed by Database administrator starting from deployment to production. On Figure 3.4, you can see CI working process, where CI provides database integration during a build.

Another essential part of CI system is **testing**. Due to the automated testing, developers are confident about changes, because it shows their correctness. Automated

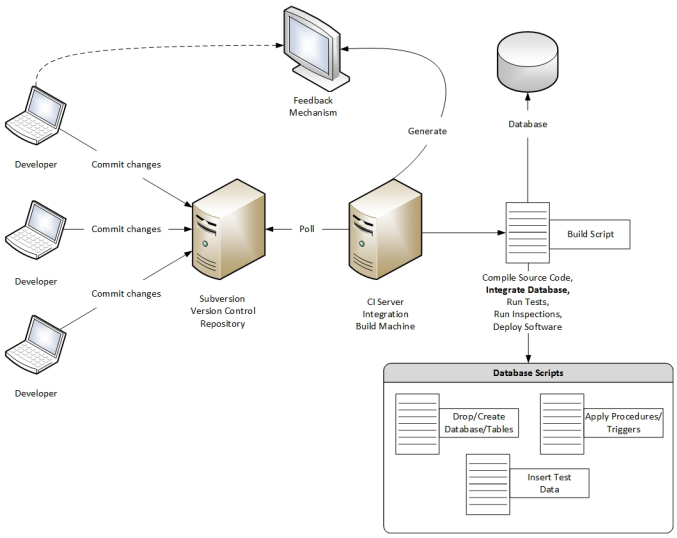

Figure 3.4 *Database integration [45].*

testing detects most of severe problems in the project. Tools such as Junit, NUnit and Xunit are considered to be most common unit testing tools. Also, most of these tools produce comprehensive reports to users and they come in form of charts, web pages or graphical reports. Unit testing checks the reliability of the code. Besides that, there are other types of tests. Component tests check the functionality and search defects in an application. System tests check functional and non-functional requirements of the whole system. Functional tests are based on the functions performed by the system and can be carried out at all levels of testing.

Inspection is the analysis of the code based on a set of predefined rules. Automated code inspection can be used to improve the quality of the software using specific rules. For instance, developers can set the rule that size of class cannot exceed more than five hundred lines of uncommented code in a project. Inspection is used for checking correctness of the written code and code duplication, additionally it is used for keeping architectural layers intact. Continuous inspection reduces the period of time between the detection and solving of the problem. Regardless of the kind of anomalies existing in the code, inspection would nevertheless notice suspicious piece of code, that has high probability of error occurrence. Despite the fact that inspection will not detect all problems, it alleviates developer's work and allow them to focus on important and complex pieces of code. The automated inspection produces result very similar to the one produced by a group of reviewers.

Typical **deployment** process is divided in six steps. First step is label repository's elements. Usually it is performed at the end of a project stage. Second step is maintaining clean environment without assumptions. It means that you need to maintain good conditions for successful integration build by removing all files, servers, configuration changes from integration machine. It would be better, if this process would be strict. Third step is making and marking a build right from the version control repository and installing it on the target computer. Fourth step is achieving successful automated tests. They are: unit test, functional test, system test and component test. They are run by developers to make sure the software is ready and can be used in the next stage of development or can be provided to customer. Fifth step is generating automated build feedback reports. At the Sixth step is possibility to roll back the build is reviewed. It means that if something would be abnormal, you can still deliver the latest viable version to user. If all of this steps have been passed, you will get successful deployment.

There is common belief that the best type of **documentation** for the source code is the ability to write a simple and short code with appropriate classes, variables and methods. CI simplifies the process of documentation creation. Tools such Maven, JavaDoc and NDoc are often used to write documentation in CI. In addition, these tools can also provide class diagrams and other information related to software development based on the committed source code to the version control repository. By using all this tools, you can get important benefits such as receiving documentation from source-code and project status in real-time.

3.3 Advantages of CI

In general, the use of Continuous Integration brings a lot of benefits to software development. Mark Duvall in his book [45] distinguishes five main properties of CI. Further sub-parts describe these properties in details and explain what they may offer to project.

One of the greatest properties of CI is **risk reduction**. It is achieved due to frequent integrations during a day. It makes easier to find problems in the project and monitor software health. Also it reduces amount of assumptions, risks of low-quality software, lack of project visibility and risk of late defect detection. In comparison with late testing, some defects or errors are detected earlier due to inspections, integrations and conduction of tests runs several times per day. In other words, bugs are found

when the code is submitted to repository. Thus more obvious control of software health is achieved. It allows to track complexity and other health attributes of project over time. Reduction of assumptions occur because of automated processes that stay unchanged by development team, which lead to decreased probability of human-induced mistake occurrence.

Continuous Integration brings **reduction of repetitive manual processes** and automates repetitive processes. These processes include code compilation, testing, database integration, inspection, deployment and feedback. CI reduces amount of repetitive manual processes and allows to save time, money and spend less effort on the project. In addition, reduction of repetitive manual processes allows team-members not to spent time to non valuable work and be focused on necessary work, also it makes possible to avoid mistakes in these processes by human.

One of the most obvious advantages of Continuous Integration is the ability to **generate deployable software** at any moment. The importance of deployable software is very high, because it is one of the most significant features for customers. Frequent deployment helps to eliminate borders between customers and developers. According to work cycle in CI detection of errors occur after making changes in a code and integration phase and at the same time all team members are informed about problems and they try to fix them as soon as possible. While in other projects without similar practices, due to late integration and testing that occurs right before delivery to customer, issues appear only in the end. It can cause delay of release due to elimination of found defects and can take a long time since it is unpredictable process, because errors are cumulative. It is very hard to remove bugs, if you have a lot of them. Besides that there is another psychological problem. This problem is called the Broken Windows syndrome. It means that people are less motivated to search for bugs and fix them, when project has a lot of them.

Continuous Integration **enables better project sisibility** and makes project more transparent to the team during development. Due to up-to-date information, it is getting easy to lead project more effectively. It brings positive effects such as effective decisions and it reveals trend tendencies. Since the integration is very frequent process, it is easier to notice all trend tendencies, that are based on successful or failed builds; overall quality is also improved due to CI.

Another main CI property that it **establishes greater product confidence** and it gives great confidence creating process of software. The knowledge of code change

impact on software makes all team-members more confident in their actions. Without frequent integrations, some team members may feel not so confident, because they are not able to see how they affect the project in general.

3.4 Negative Aspects of CI

Though much was said about benefits of CI, there are some negative aspects of CI. Duvall presents in his book combination of different circumstances [45].

Firstly, it is prejudice. Often CI system is considered to be increasing overall cost of the project. However, present projects have phases such as integration, testing, inspection, and deployment despite the CI system absence. It is much easier to control reliable CI system than to try to manage manual processes. Curiously enough, the CI system is considered unnecessary by members of complex projects where CI would be very useful. Projects members think that there is too much extra work for setting up CI.

Secondly, it is believed that the implementation of CI system on current projects would cause many changes in many processes. The most effective method of CI implementation is an incremental approach. It implies that builds and tests occur only one time per day and eventually increase the frequency as everyone gets familiar with it.

Thirdly, wrong application of CI practice may lead to many failed builds. Generally, it happens when developers do not create private builds before submitting their code to repository. For instance, developer can submit corrupted files that lead to tests failure.

Fourthly, it is additional expenses for buying hardware for CI, also it may be required to pay for software, if it is not open-source. In comparison with future expenses for finding and fixing problems in last phases of software development, this costs are reasonable.

According to Cauldwell [43], other barriers why every team do not practice Continuous Integration already, firstly because it is hard to establish a solid CI practice and it needs good technical experience. Secondly, it is difficult to persuade developers to be involved, because of habitual style of working, into the idea that it is important and that CI is good idea.

4. THESIS APPROACH

In previous chapters concept of Continuous Integration was introduced and clarified. As stated in the first chapter, the main research problem of this study was to show comparison of continuous integration tools. As reminder, underlining research questions were presented in the beginning as follows:

Which criteria and requirements are appropriate for examining CI tools?

Which tool is good as entry-level tool for getting acquainted with CI area?

Which tool satisfies mostly to our criteria and requirements?

According to "Conducting online surveys" book by Valerie M. Sue [61], research process can be divided into three types: exploratory, descriptive, and explanatory research. Whereas, individual study can be consisted of different purposes and includes two or all three categories of research types. For instance, research projects can be explore and describe; explore, describe and explain; or describe and explain [53].

Exploratory research aims to explore issue or a topic. It is used when it is needed to identify a problem, clarify concepts, define the involved issues or reach a greater understanding of an issue. Exploration can begin with a literature search, a focus group discussion, or case studies. Data from exploratory studies tends to be qualitative. Examples include brainstorming sessions, interviews with experts, and posting a short survey to a social networking website. Exploratory research is a significant first stage in the research design process.

Descriptive research is used for describing things, events, or situations [54]. The difference between exploratory and descriptive research is that, when using descriptive technique, you have a clearer idea of what is needed and you are looking for answers to more clearly defined questions. Descriptive studies might tell you, for

example, the size of a market, the structure of the market, developments over time, attitudes of particular groups of people and etc. A study of this type could start with questions such as: "What similarities or contrasts exist between A and B?" Where A and B different tools which is used for same purposes. Such descriptive comparisons can produce useful insights and lead to hypothesis-formation.

The primary purpose of explanatory research is to look for explanations of the nature of certain relationships. The data are quantitative and almost always require the use of a statistical test to establish the validity of the relationships. Explanatory research addresses the why questions: Why do people choose brand A and not brand B? What might explain this? This research answers for these types of questions allow to rule out rival explanations, come to a conclusion and helps in developing causal explanations.

Table 4.1 *Differences between qualitative and quantitative methods.*

	Quantitative method	**Qualitative method**
Applied in following techniques	Exploratory, descriptive and explanation	Exploratory and descriptive
Questions and responses	Who, what, when, where, how many? Relatively superficial and rational responses Measurement, testing and validation	Why? Below the surface and emotional responses Understanding, exploration and idea generation
Sample size	Relatively large	Relatively small
Data collection	Not very flexible Interviews and observation Standardised More closed questions	Flexible Interviews and observation Less standardised More open-ended questions
Data	Numbers, percentages, means Less detail or depth Nomothetic description Context poor High reliability, low validity Statistical inference possible	Words, pictures Detailed and in-depth Ideographic description Context rich High validity, low reliability Statistical inference not possible

Additionally, Hague et al. [48] tell that researches can include such data collection methods as qualitative and quantitative. A useful way to distinguish between the

two methods is to think of qualitative methods as providing data in the form of words and quantitative methods as generating numerical data [53]. Following table 4.1 clarifies these data collection and shows theirs differences [53].

Consequently, our literature research would be multi-dimensional and it is consisted of explorative, descriptive and explanation techniques with qualitative and quantitative collection of data. Generally, collecting of relevant data for thesis is based on secondary sources (desk research). It means that for survey is used information which was previously printed or published for other purposes and publicly available. It includes published research reports, surveys from interviews, books, web-sources and other existing material.

As concerns main research problem, this survey can be considered as explore, describe and explain with exploiting qualitative and quantitative data gathered by desk research. Before making comparison and evaluation of CI tools, we need to give answer for first question. In order to answer, it would be used explorative technique with collected qualitative data from secondary sources for finding criteria or desirable qualities. In next step, for selection of tools was used same research method, but with quantitative data. Concerning second and third questions, our strategy is based on descriptive and explanatory technique. We would analyze gathered information from previous steps and find new one from secondary sources such as official web pages, journals, web articles, books, forums, question-and-answer websites. Finally, we collect qualitative data from questionnaires on one of the most popular software review platforms with audience more than 400000 users [1] and would use exploratory and Descriptive types of research. Questionnaire consist of following questions:

1. How likely is it that you would recommend this particular tool to a friend or colleague?
2. What do you like best?
3. What do you dislike?
4. What business problems are you solving?
5. For which purposes do you use this particular tool?

According to analysis of information our basic principle consists of the following steps. Firstly, we read through all gathered information from questionnaires, interviews, etc. Beside of that, we review aims concerning research questions, since it helps us to organize data and focus on analysis. Further, we categorize information according to each measure associated with description, statements, similarities, thoughts and opinions into similar categories, e.g., average description, strengths, weaknesses, repetition frequency of statements, etc. Next, we attempt to identify patterns, or associations and causal relationships in the categories for each CI instrument. Subsequently, we have the intention to summarize and bring all output from previous steps to single whole and try to show differences and general picture of researched tools. Finally, we design comparison table with researched tools and gathered data.

5. RESULTS

In today's world because of the fast grow and big variety of choices in the Information Technology market, selection of right tools is becoming a very hard task. Frequently, criteria are not clear and they are not defined at the beginning of tool choosing process for Continuous Integration deployment. It is important to find out what qualities of CI are required when choosing the tools. Due to this problem, in this chapter, we try to carry out the research based on literature and web forums of developers to determine which criteria, requirements and features of CI tools are common, optimal and important despite of differences between projects. First section of this chapter describes main requirements in evaluating CI tools based on selection of literature, including Duval's book, web-forums and some Internet platforms, where people share opinions. Goal of second section to present our description of Continuous Integration tools based on gathered information from more than one hundred reviews, forums and official tool's pages. As a consequence of first two sections, in third section our aim to present a comparison table.

5.1 Choice Criteria for Optimal Continuous Integration Tool

Paul Duvall, in his book "Continuous Integration: Improving Software Quality and Reducing Risk" [45], states that the best CI tool should both save time and costs of a project. Also good CI tool must serve the team of developers in many projects, rather than in one. Although it is not always like that, he said that tool selection is not a lifelong contract, whereas it just means that tool should fit your needs well and you should know how to work with it. He also points that due to some circumstances, the team of developers might want to change a tool in middle of the development process. According to his past development experience, he had to change tools in the middle of the process of development and this change was justified.

Obviously, **functionality** is one of the most significant factors in choosing CI tool.

It includes following features.

- Build execution is base functionality feature of CI tool. Duvall claims that real CI system must be polling-driven, whereas schedule-driven with predetermined schedule for build execution is not true Continuous Integration tool. However, there exists another type of build execution - event driven. It means that build is performed after any changes in repository are made. It sounds better, but it can require additional time configuring version control repository, while in a polling-driven system it is not required.
- Undoubtedly, maintaining the version control system, which is used in a development, is an important factor. We should pay attention to factors, such as how CI tool interacts with methods of repository use. At least, CI server have to recognize changed files and their versions.
- Feedback is key factor of CI. Different types of feedback are supported by CI; you just need to consider system with appropriate options that fit in a project.
- User interface is not essential criterion for CI tool, although should be taken into considerable, because good designed interface can save your time at early periods of time while working with CI tool. User interface can be provided in different forms; for instance, it can be a web application interface or some other interface.
- Different CI tools include security features built-in in their system. It means that these tools include authentication and authorization, which allow control access to results review and a configuration modification.
- Extensibility. This term implies that possibility to extend functionality can be significant factor in choosing CI tool.
- Build tool integration.
- Build labeling.

Compatibility is a criterion that defines how well CI tool is integrated with other elements of development process. You need to find answers to series of questions regarding compatibility. Firstly, you have to ensure that current build configuration by CI tool is supported. Secondly, installation of additional software for CI tool

may reveal some difficulties. Sometimes tool can require installation of additional software, for instance it can be a new execution environment. Thirdly, language compatibility between tool and developed project is another problem. There are a big variety of CI tools to choose. Some of them are created for working with specific programming languages while others like CruiseControl can work with any programming language. If you do not know what kind of projects would be developed in the future, it would be better idea to start from CI tools, which supports many programming languages. Nevertheless, if you are sure of your requirements in advance, it would be better to choose tool that is designed for one language, because you do not need support of others.

Clearly, **reliability** is important, because CI tool with a long period of existence on the market and good reputation would be more reliable choice than a tool which is still in beta-version state and just appeared at the market. Another crucial factor is communities related to this tool. If the size of community is big with large amount of developers and users, it can indicate that there are a lot of materials and answers to questions, and it would be easy to find solutions to problems, which might appear during a period of a tool usage.

In comparison with reliability, **longevity** is a factor that concerns future of the tool, while reliability shifts attention to past and present of tool's lifespan. In addition, as with the previous factor, the size of community and popularity between users and developers is important. Although, it is not obvious, longevity is higher in tools with open source code, since community of users is constantly kept up-to-date them. Furthermore, good CI tool with diverse functionality is used longer than a tool with poor functionality.

Another considerable criterion is **usability**. If tool is easy to configure and use, it means a lot. Using different CI tools for a long period of time will help you to realize that it is important. Mostly, you can reveal one common difference in usability among diversity of tools, which is new project configuration process.

Money is always relevant, it is a reason why **free, open-source or commercial** criterion is shown. Nowadays, there are a lot of free and commercial software available on the market. Commercial tool is not always better than free one; comparison and consideration of all factors before choosing a tool should be performed.

5.2 Continuous Integration Frameworks

At this point, we have explored different web-sources. Mainly, we studied Google trends, forums, Stackshare and G2 Crowd platforms. Therefore, we have selected following six most faced and popular tools for today.

5.2.1 Jenkins

Jenkins is cross-platform and leading award-winning open-source Continuous Integration server based on Java platform, what main goals are building and testing software continuously and monitoring executions of externally run jobs [21]. It can be said that it is most popular CI tool. Open-source and ability for extension are main factors of success. It was created by Kohsuke Kawaguchi in 2004 and, initially, it was called Hudson, but in 2011 it was renamed to Jenkins because of disputes with Oracle [39]. The tool simplifies the process of integration of changes in to the project and delivery of fresh builds to users. There are already more than 1200 plugins available, that extend Jenkins functionality [21]. Community of Jenkins consists more than 100 thousands users and it grows very fast, as for December 28 it was around 127,000 [29].

Jenkins' interface is represented as series of web pages. User interface of Jenkins is shown on the Figure 5.1.

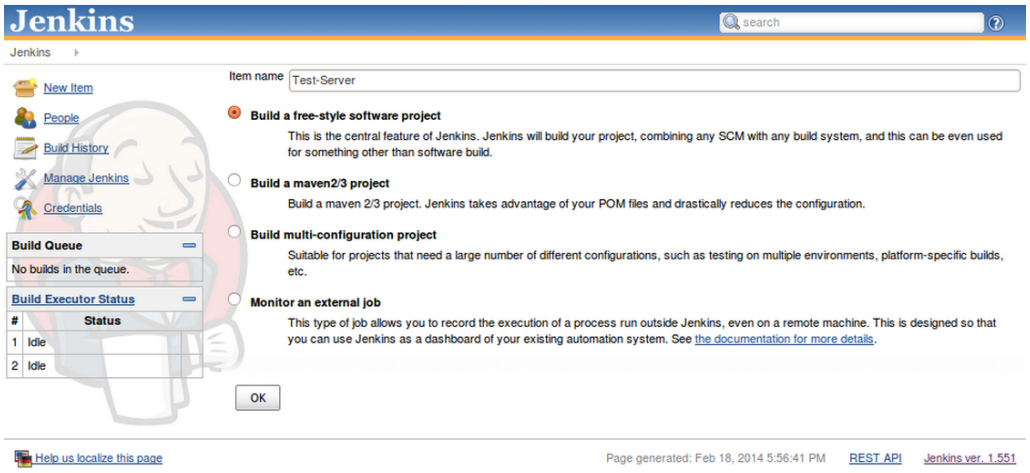

Figure 5.1 *Jenkins main screen*

Jenkins takes first [28] and second [6] places among most popular CI tools in G2 Crowd and Stackshare portals for today. Also, according to ZeroTurnAround survey,

Jenkins holds dominant position with 70 percent among other tools on the market in 2013 [19].

Jenkins is used by variety of companies, most notable of them: Facebook, Sony, Yahoo, Ebay, Netflix, Tumblr, Linkedin, Vine, Hootsuite, Etsy [20].

According to Jenkins official web-site [21], Jenkins offers the following features:

1. Easy installation.
2. Easy configuration: Jenkins can be configured entirely from their web graphical user interface.
3. Rich plugin ecosystem: Jenkins integrates with virtually every service control manager or build tool that exists.
4. Extensibility: Most parts of Jenkins can be extended and modified, and it is easy to create new Jenkins plugins. It allows you to customize Jenkins for your needs.
5. Distributed builds: Jenkins can distribute build/test loads to multiple computers running under different operating systems (OS X, Linux, and Windows).

Other Jenkins features [28]:

- Change set support
- Permanent links
- RSS/E-mail/IM Integration
- After-the-fact tagging
- JUnit/TestNG test reporting
- File fingerprinting

Based on different forums and resources, where users provide their opinions and reviews, we can construct general list of Jenkins CI properties, by showing pros and cons. Most Stackshare users emphasize such Jenkins features [20]: firstly it can be

hosted internally (211 votes); secondly, it is free and open-source(169 votes); then it is great to build, deploy or launch anything async (125); fourthly they underline that it has big variety of plugins with good documentation (90) and there are tons of integrations (71). According to G2 Crowd [22], most of Jenkins users claim that they are very happy with this product. Most of them say that Jenkins is very powerful tool with easy initial setup and straightforward configuration. Users denote that the tool is open-source. The process of data deployment is well visualized. Jenkins supports several service control systems, including popular GitHub and SVN. There is excellent support of Maven and Java. Community of Jenkins users is very large and active. It means that there is a wide array of documentation which can be helpful in resolving some issues and there is impressive amount of plugins for diverse and specific sets of tasks. Most of the user's reviews lead to conclusion that Jenkins is a valuable member of development life cycle, and it savesd them countless work-hours. It takes time to set it up properly and there is an adoption barrier. If you have finished the whole system setup, it will meet your requirements and needs.

On the other side, it has negative points. Jenkins might be quite resource hungry software, in case you have installed a lot of plugins and you have a lot of build jobs [15]. According to G2 Crowd [22], users tend to think that interface of Jenkins is poor and old-fashioned. Also, some of them argue that it is not intuitive, because sometimes it forces a user to makes so many clicks to find desirable information. As it was said before, Jenkins is easy in initial setup, whereas to use complex plugins is not as easy and needs some pre-configuration in administration. For some of users, it was hard to deal with Jenkins documentation. Process of updating can be painful, because plugins can crush after an update and it needs time for restoring a system to functional state. In addition, the amount support is less in comparison with commercial products.

5.2.2 Bamboo

Bamboo is a commercial Continuous Integration tool developed by Atlassian, but it is free for open-source projects. It connects automated builds, tests and releases together in a one workflow. Bamboo supports any languages and other spread technologies like AWS CodeDeply, Docker, Amazon S3 buckets, Maven, Ant, Git, Mercurial, and SVN. Bamboo finds new branches in Git, Mercurial, and SVN repositories, and then automatically applies the main line's CI scheme to them. Bamboo provides deployments with the first-class treatment. Integration with Amazon S3,

ready-to-use Docker agents and AWS CodeDeploy and Docker tasks make the deployment process faster and easier. Bamboo provides good integration with Jira with ability to extend even further. Bamboo can be extended by plugins in the same manner as Jenkins. Build notifications can be modified based on the type of event, and get sent by email, instant messages, RSS, or pop-up windows. Bamboo supports easy migration from Jenkins [24].

Following companies use Bamboo: National Public Radio, LeapFrog, Williams-Sonoma, Cisco, BMW, Kaiser Permanente, NASA [24].

According to G2 Crowd [22], Bamboo users say that it is very flexible tool with good Atlassian products integration, like Jira and Fisheye. Bamboo documentation is good and detailed. They say that Atlassian provides excellent support. One of users, who have used Bamboo for seven years, says that each time when he asked for the support in Atlassian, they were very helpful. It is easy configurable, it supports many popular programming languages and Bamboo provides multiple notification methods. Bamboo works fast and efficiently when configuration and environment are properly set up. On the other hand, it has negative moments; if the project is not open-source, Bamboo will be free only for first seven days. Some of reviewers say that user-interface is not good and they hope that Atlassian will focus more on mobile applications. First work experience with Bamboo can be complicated. For some of users setup process was difficult and not clear enough to understand.

5.2.3 CircleCI

CircleCI provides easy setup and maintenance without any difficulties [8]. It eases development for software teams in terms of building, testing and deploying fast and systematically through different platforms. CircleCI supports all types of software tests including web, mobile and container environments. CircleCI makes the process of Continuous Integration simple and easy for any company [7].

Following companies use CircleCI: RedBull, Kickstarter, Stripe, Percolate, Teespring, BOOTH, Couchsurfing, Harvest, Vusay and Gazelle [8].

CircleCI integrates with GitHub, Heroku, Amazon EC2, Rackspace Cloud Servers, Joyent Cloud, Engine Yard Cloud, Nodejitsu, AppFog, dotCloud [8].

- Based on Stackshare quiz, following attributes of CirleCI are the most empha-

sized for Stackshare's community members [8]:

  - Github integration, fast builds, easy setup, competitively priced, slack integration, great customer support, docker support.

CircleCI Features [8]:

- Quick Setup. Process of setting up is around 20 seconds. CircleCI detects test settings for a wide range of web apps with a single click, then settings on CircleCI servers are sat up automatically.
- Deep Customization. CircleCI settings are very flexible, allowing to easy set up almost anything required.
- Debug is made with ease. When tests are corrupted, CircleCI helps to fix them. It auto-detects errors and provides great support.
- Smart Notifications. It can intelligently notify via email, Hipchat, Campfire and others sources. Notifications contain only relevant information.
- Fast support. CircleCI respond to support requests very fast, users do not need to wait more than 12 hours for a response.
- Automatic Parallelization. CircleCI can automatically parallelize tests with up to 4 parallel flows via multiple machines.
- Continuous Deployment, build artifacts, clean build environments, GitHub integration, fast tests, free Open Source support [12].

According to G2 Crowd [22] reviews, CircleCI is intuitive and extremely helpful tool, even one of the reviewers claims that it is faster than Travis and simpler than Jenkins. They characterize it as quick and easy to set up and benefitial to development processes. Moreover, one reviewer found that their documentation is good and when he had questions, CircleCI support helped him very fast. There is a 14 day free trial period to try Circle CI, since it has time to see whether it meets all team requirements or not. The user interface is very clean and easy to understand. One of reviewers emphasized increased speed of parallel tests. There are premade configurations with good integration to GitHub. Notifications are timely and helpful, they can be sent by email, chat and webhook notifications. On the

other side there are some disadvantages. For instance, there were some cases where tests were broken because of CircleCI upgrades. Although, it is a small complaint, but it would be better if it will be possible to define environment variable for all projects. Sometimes tests cannot be passed because some extra configuration is required. Also, CircleCI sends notifications to many emails each time when a push is performed. One of reviewers said that user experience was not so comfortable and it could be improved. In addition, it is not always updated with latest java development kit versions.

5.2.4 Codeship

Codeship is a Continuous Integration and deployment tool for web applications, including php, rails, java, python or go apps. It runs automated tests and configures deployment after changes are made in a repository. It keeps managing and scaling the infrastructure, thus it allows developers to be able to test and release more frequently and receive fast feedback for improving the process of a development software creation [10].

- Based on Stackshare quiz, following attributes of Codeship are most emphasized by Stackshare's community members [10]:
  - Easy deployment, simple setup, GitHub, Bitbucket and Slack integration, intuitive and great UI, customer support, easy to get started.

Codeship's features [10]:

- Running automated tests.
- Codeship is easily to set up with GitHub or Bitbucket and trigger automated tests with a simple push to repository.
- Setting up powerful deployment pipelines that let to deploy with ease multiple times per day.
- Intelligent notifications and integration keep team informed.

Following companies use Codeship [10]: Bia2, Treehouse, Product Hunt, Bannerman, Soylent, Podigee, Pipedrive, miDrive. Furthermore, it supports integration with GitHub, Slack, Heroku, Bitbucket, Amazon EC2, HipChat and Capistrano.

According to G2 Crowd [22], almost all reviews point out that setup is very quick and it easy to get started with. They say that in Codeship all things are much easier and process of deploying much simpler than in Jenkins. It is easy to install with GitHub and easy to configure tests, also it is very convenient to deploy to Amazon Web Services. Some of reviews imply that user experience is very good in Codeship. People liked how shared setup, testing and deployment are implemented in Codeship. Most of them emphasize that process of working with Codeship is simple and user-friendly. People describe user interface as intuitive and elaborate, because all tools are powerful and work consistently and it does not take much time for user to get familiar with GUI. For instance, if build fails, developer knows that it the problem is with a code or a test and not with the platform. This technique allows to troubleshoot and determine issues much faster. Some reviews laid accent on the ability to support Secure Shell in a build environment. Some of them recommend everyone to try out Codeship, especially small teams with tight budget since Codeship offers a reasonable functionality in their free package.

Nevertheless, it has negative properties [9]. In some reviews, users suggested to add support of storing artifacts. Then, there are slight problems with notifications. Most of reviews tell us that there is lack of documentation and it should be extended, because it bothers them, especially in comparison with Jenkins. In addition, some people say that price is high and it would be better if they add more plugins.

5.2.5 TeamCity

TeamCity is a multifunctional Continuous Integration server, which is ready to work right after installation. It supports many version control systems, authentication, deployment and testing out of the box. TeamCity is extensible, for many operations it does not even need to support Java. It is easy to setup and it is free for small teams and open source projects. TeamCity is used by following known companies [27]: Apple; Ebay; Jvm stack; Yammer; Stack exchange; NationBuilder; Stack Overflow.

Most relevant Team City's features are described further.

Instant feedback on issues and test failures during a build progress. It does not re-

quires user to wait for the end of the building to be able to see errors in compilation or test failures.
Ability to run compilation and testing processes of modified code without a commit to the version control system, directly from the IDE.
Simplified setup - it allows to create projects from just a VCS repository URL.
Run multiple builds and tests under different configurations and platforms simultaneously.
The hierarchical structure of projects, making it easy to set up rights and server configuration.
Big statistical reports on results of the builds with information about build duration, success rate, code quality, and custom metrics.
Great visual project representation. It tracks any changes made by any user in the system, filters projects and chooses style of visual change status representation.
Management of common resources allows to get easily access to shared databases, test devices, etc.
Configurable conditions of failures in a progress of build based on a variety of metrics, including number of failed tests, number of uncovered classes and modules, additionally metrics excluding the degradation of the quality of code.
Unique features for server support in appropriate conditions: built-in cleaning build stories, reports about occupied disk space and reports about health of the server.
Support of mixed authentication which allows to use different authentication methods (LDAP, Windows Domain, built-in) at the same time.
Excellent integration with version control systems: support multiple systems to a single project, feature branching for the Mercurial and Git, advanced rules to start building based on changes in version control systems.
Roles and user groups allow to set up an easy access to the server for all users.
Support of Ruby and XCode projects, intuitive web-interface, service message support, more than 100 free plug-ins available, easily-to-extend TeamCity functionality and new integrations with Java API.

According to G2 Crowd reviews [17], the installation of TeamCity is very easy and quick. Moreover, user interface of TeamCity is simple and easy to get familiar with and provides large variety of options. TeamCity reports project status and relevant information for various users and even project stakeholders. TeamCity includes documentation with a lot of examples and tutorials for different cases. In addition, users say that it is easy find answers on the Internet, including such topics as the platform setup, due to the fact that TeamCity has been present on the

market for a long time and gained popularity among users. There is a long but very straightforward process of setting up builds, which makes possible to modify options, like making a schedule for a build trigger. One of the most popular features in TeamCity is an automation of different aspects of the software development life cycle. By automating development process users are able to gain more productivity. The process allows to decrease deployment time and keeps track on easily configurations of deployment process. Build configuration is very flexible. It includes building source code with difference targets, shell scripting etc. Furthermore, reviews present TeamCity as a good product for development team of any size. Moreover, it suits to the companies with different environments and services, since it can be extended to fit the projects. However, it has some shortcomings [17], some of reviewers say that they were not satisfied with documentation and process of setting up and upgrading process was also difficult. One of the reviewers dislikes explanation of software specific terminology that mostly have to search the meanings of terms in the Internet. It is said that tooltips can bring more meaningful and immediate feedback than opening windows with big amount of information.

5.2.6 Travis

Travis CI is a web-based Continuous Integration server for open source and private projects closely integrated in GitHub. In the beginning, Travis CI supported only Ruby-on-Rails, but currently it supports almost all kinds of languages and platforms. The main idea of Travis is an simple integration with GitHub and adding a single file and a post-push webhook triggers in the hosted virtual machine [16].

Travis CI is used by many companies [30] like TimeHop, JVM Stack, 500px, Fluxible, Moz, Heroku, Techstack, Code School.

Based on Stackshare quiz [30], the following attributes of Travis CI are the most emphasized for Stackshare's community members:

- GitHub integration
- Free for open source projects
- Easy to get started
- User-friendly interface

- Automatic deployment
- Tutorials for each programming language

Based on G2 Crowd reviews [31], Travis CI can be characterized by following features. Firstly, it is free for open source projects. Then, one of the main features that it includes the most popular programming languages and server technologies. Travis CI has clear configuration specification and it has wide user community. It provides fast support and process of updating software in the stack. Furthermore, another good feature is seamless integration with GitHub and it is even not necessary to go into Travis CI's website. Every pull request and commit presents current build status in GitHub. Additionally, it saves time by running large test suites on a remote system that allows developers to continue the work without stopping for waiting their own workstations to run the test. The user interface is very responsive, most users say that it is easy to use Travis CI and to do the work, configuration, easily monitoring builds. Travis CI allows to deploy easy-to-perform tests concurrently. In addition, it is free to have one private repository for students. But it has negative moments. Several reviewers noted that in some cases setup process took significant amount of time and it was not easy to understand all documentation. It can be not stable sometimes. During compilation or running tests, it can freeze for unknown reasons requiring manual interruptions. Most of the time this problem does not happen but it occurs occasionally. Some users say that they would like to add SSH to the build machine to debug a build failure, then to have an ability to manually trigger a new build and to re-arrange or re-prioritize the builds. There is no good support for Docker. Setting up the .travis.yml files can be tricky and require many excessive commits to get a build working properly.

5.3 Decision Matrix for Continuous Integration Frameworks

This section presents comparison table 5.1 with researched Continuous Integration tools. Besides of criteria presence, we try to show relation of some tools to others by amount of ticks. Whereas tick with a star denotes limited functionality in this type of criteria.

Table 5.1 *Decision Matrix for Continuous Integration Frameworks.*

Criteria	**Jenkins**	**Bamboo**	**CircleCI**	**Codeship**	**TeamCity**	**Travis**
GitHub	✓	✓	✓	✓	✓	✓
Other repositories	✓	✓	✓	✓	✓	×
Multi-Language	✓	✓	✓	✓	✓	✓
Feedback	✓	✓	✓	✓	✓	✓
Usability	×	✓	✓	✓	✓	✓
Security	✓	✓	✓	✓	✓	✓
Extensibility	✓✓✓	✓✓	✓	✓	✓	✓
Compatibility	✓✓✓	✓	✓	✓	✓	✓
Reliability	✓✓✓	✓	✓	✓	✓	✓
Longevity	✓✓✓	✓	✓	✓	✓	✓
Commercial	×	✓	✓	✓	✓	✓
Totally free	✓	×	✓*	✓*	✓*	×
Trial	×	✓*	✓*	×	×	×
Free for open-source	✓	✓	×	✓*	✓*	✓*
Support	×	✓	✓	×	×	×
Easy entry period	×	×	✓	✓	×	×
Flexibility	✓	×	×	×	×	×
Community	✓✓✓	✓	×	✓	✓	✓

The table uses eighteen criteria to show relation and comparison for six selected CI tools. It can be clearly seen that the Jenkins has biggest amount of ticks than other five instruments. Furthermore, we can see that there are not significant differences in criteria, whereas comparison can be provided based on relation to each other. For instance, Jenkins provides biggest extensibility in comparison with other tools, due to this fact, it was assigned three ticks.

6. DISCUSSION

There were not found surveys that were attempting to compare existing continuous technologies and bring them together. In our section 5.1, after analyzing market of CI users we emphasized main qualities and criteria for considering existing tools. Furthermore, in section 5.2, we have explored besides of official pages of CI tools also huge number of reviews and summarized gathered information to one description. At this section, we have arrived to the point where we will begin presenting our solution and choice among alternatives.

Consequently, the choice was not so easy since Jenkins and Codeship provides a lot of valuable services and important features. Although, Codeship doesn't have so many disadvantages, the final decision came to Jenkins. Jenkins was preferred over other tools since it allows to safe budget and provides beginner-friendly set of CI tools. Firstly, it is a free open-source project. This factor was one of the most common and important among reviewers. Secondly, Jenkins provides great flexibility to so many different development methodologies and it has all of the features that a developer might need. There are more than 1200 plug-ins that allow user you to assemble desirable configuration. Jenkins also has the biggest community which is very motivated to improve Jenkins and very active. Fourthly, it is good start for understanding which criteria is most appropriate and crucial, and after that it would be easier to change or choose new CI tool, on the contrary continue to choose Jenkins. In comparison with Jenkins, Codeship has better UI and works better in case when you have installed huge amount of plugins to Jenkins because it slows it down. Additionaly, Codeship has better support service. Users say that in case they had any issues and need any support they always got it incredibly soon. Despite the fact, Codeship provides so many good service, it has a free plan with reduced functionality for open source projects. It is still advanced version of CI tool, whereas Jenkins is entry level tool in the area. Also, another crucial factor why Jenkins was chosen, it has opportunity to find almost any answers for any issues. Lifespan of Jenkins is the biggest in this area, whereas another old tool, like Cruise Control, is

faded. Based on our research, we can argue that Jenkins beside that is perfect option as entry tool, it can be claimed that it is good as usual tool and have tremendous functionality. Jenkins definitely fits to majority requirements and criteria described in section 5.1 and meets our thesis research questions.

7. CONCLUSION

The goal of this thesis, as it was stated in the first chapter, is to clarify the best instruments, show the overview of CI tools and make a comparison. We expect that after reading a thesis, results of survey will help making a decision of the tool to some developer or at least to get some comprehension. We feel that it was taken most latest and actual Continuous Integration tools on the market and used for survey best available sources and researches.

After answering to thesis questions, we can express our thoughts, that you cannot select ideal CI tool, as most effective and universal, before you will try at least one of them, since each considerable case has own conditions and requirements. Nevertheless, as it was said in previous chapter, Jenkins would be great choice as best entry and flexible tool in an acquaintance with CI area and for a long work, also it is good for defining specific requirements.

This thesis has shown that future work on this topic has great potential. The work on a thesis gave us a lot of ideas. For instance, after deploying CI server, we can make research based on ideal configuration and what kind of other tools is most suitable in a work. In addition, it is possible to go in direction of developing some specific plugins to Jenkins, if it is not available for some case. Then, it is very easy to find other possibilities for future work into the CI topic.

In conclusion, this thesis project can be helpful for those who are looking a short guide, where would be suited basic knowledge for starting work with CI. Since, this thesis work allows to dive into CI process and learn in parallel from where it takes origins. As a result, this research gives to the reader possibility to get acquainted with most new popular tools, and get first imagination about CI without wasting time for searching not always necessary information.

BIBLIOGRAPHY

[1] About g2 crowd page. https://www.g2crowd.com/about/. [Online; accessed Jan. 2016].

[2] Agile practices. http://www.accegile.com/how-we-work/agile-practices.html. [Online; accessed Jan. 2016].

[3] Agile: The world's most popular innovation engine. http://www.forbes.com/sites/stevedenning/2015/07/23/the-worlds-most-popular-innovation-engine/#76df6aed2d4c. [Online; accessed Mar. 2016].

[4] Agile vs. waterfall website project management methodologies. http://www.intechnic.com/blog/agile-vs-waterfall-website-project-management-methodologies/. [Online; accessed Mar. 2016].

[5] Agileversuslean. https://http://martinfowler.com/bliki/AgileVersusLean.html. [Online; accessed Jan. 2016].

[6] Build automation software list. https://www.g2crowd.com/categories/build-automation/products. [Online; accessed Jan. 2016].

[7] Circleci - continuous integration and deployment. https://angel.co/circleci. [Online; accessed Jan. 2016].

[8] Circleci continuous - integration for web apps. http://stackshare.io/circleci. [Online; accessed Jan. 2016].

[9] Codeship. https://www.g2crowd.com/products/codeship/reviews. [Online; accessed Jan. 2016].

[10] Codeship - continuous integration and delivery made simple. http://stackshare.io/codeship. [Online; accessed Jan. 2016].

[11] Continuous everything in devopsâwhat is the difference between ci, cd, cdâ? https://www.accenture.com/us-en/blogs/blogs-continuous-everything-devops. [Online; accessed May. 2016].

[12] Continuous integration and deployment. https://circleci.com/. [Online; accessed Jan. 2016].

[13] Continuous integration and quality. http://www.summa-tech.com/blog/2013/03/24/continuous-integration-and-quality. [Online; accessed Jan. 2016].

[14] Continuous integration in devops. https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html. [Online; accessed May. 2016].

[15] Continuous integration servers in comparison. http://www.warp1337.com/content/continuous-integration-servers-comparison-shoot-out. [Online; accessed Jan. 2016].

[16] Devprod report revisited: Continuous integration servers in 2013. (http://zeroturnaround.com/rebellabs/devprod-report-revisited-continuous-integration-servers-in-2013/). [Online; accessed Jan. 2016].

[17] Intelligent ci server. https://www.g2crowd.com/products/teamcity/reviews. [Online; accessed Jan. 2016].

[18] Is agile or waterfall the best? the answer is not binary! http://www.uxcconsulting.com.au/media/UXCConsulting_WhitePaper_IsAgileorWaterfalltheBest_web.pdf. [Online; accessed Jan. 2016].

[19] Java tools and technologies landscape for 2014. http://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-for-2014/12/. [Online; accessed Jan. 2016].

[20] Jenkins - an extendable open source continuous integration server. http://stackshare.io/jenkins. [Online; accessed Jan. 2016].

[21] Jenkins. an extendable open source automation server. https://jenkins-ci.org/. [Online; accessed Jan. 2016].

[22] Jenkins is an award-winning application that monitors executions of repeated jobs, such as building a software project or jobs run by cron. https://www.g2crowd.com/products/jenkins/reviews. [Online; accessed Jan. 2016].

[23] Lean vs. agile. http://www.onedesk.com/2013/07/lean-vs-agile/. [Online; accessed Mar. 2016].

[24] Official bamboo page. https://www.atlassian.com/software/bamboo/. [Online; accessed Jan. 2016].

[25] Review of methodology of scrum. http://ossystem.com.ua/de/blog/roundup_methodology_scrum. [Online; accessed Jan. 2016].

[26] Sdlc - waterfall model. http://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.html. [Online; accessed Mar. 2016].

[27] Teamcity is an ultimate continuous integration tool for professionals. (http://stackshare.io/teamcity). [Online; accessed Jan. 2016].

[28] Top 10 continious integration tools. http://stackshare.io/continuous-integration. [Online; accessed Jan. 2016].

[29] Total jenkins installations. http://stats.jenkins-ci.org/jenkins-stats/svg/total-jenkins.svg. [Online; accessed Jan. 2016].

[30] Travis ci - a hosted continuous integration service for open source and private projects. http://stackshare.io/travis-ci. [Online; accessed Jan. 2016].

[31] Travis ci - test and deploy with confidence. https://www.g2crowd.com/products/travis-ci/reviews. [Online; accessed Jan. 2016].

[32] Understanding the pros and cons of the waterfall model of software development. http://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/. [Online; accessed Jan. 2016].

[33] The waterfall model explained. http://www.buzzle.com/editorials/1-5-2005-63768.asp/. [Online; accessed Jan. 2016].

[34] We need more agile it now! http://www.drdobbs.com/architecture-and-design/we-need-more-agile-it-now/240169361?queryText=Release%2Bmanagement. [Online; accessed Mar. 2016].

[35] Why we should rethink the agile manifesto: Software is everywhere. http://www.jamasoftware.com/blog/rethink-agile-manifesto-software-everywhere/. [Online; accessed Mar. 2016].

[36] S. K. Amanpreet Kaur Boparai, "Process of moving from traditional to agile software development," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. 1, pp. 586–591, 2015.

[37] K. Auer and R. Miller, *Extreme programming applied.* Addison-Wesley Indianapolis, Indiana, 2002.

[38] S. Balaji and M. S. Murugaiyan, "Waterfall vs. v-model vs. agile: A comparative study on sdlc," *International Journal of Information Technology and Business Management*, vol. 2, no. 1, pp. 26–30, 2012.

[39] A. Bayer, "Hudson's future," 2011.

[40] K. Beck, "Extreme programming: A humanistic discipline of software development," in *Fundamental Approaches to Software Engineering.* Springer, 1998, pp. 1–6.

[41] K. Beck and C. Anders, "extreme programming explained: embrace change addison-wesley," *Reading, Ma*, 1999.

[42] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, *et al.*, "The agile manifesto," 2001.

[43] P. Cauldwell, *Code Leader: Using People, Tools, and Processes to Build Successful Software.* John Wiley & Sons, 2008.

[44] W. Commons, "Wikimedia commons," [Online; accessed Mar. 2016].

[45] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk.* Pearson Education, 2007.

[46] M. Fowler and M. Foemmel, "Continuous integration," *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 2006.

[47] B. Grady, "Object-oriented analysis and design with applications," 1994.

[48] P. N. Hague, N. Hague, and C.-A. Morgan, *Market research in practice: How to get greater insight from your market.* Kogan Page Publishers, 2013.

[49] M. Httermann, *DevOps for developers.* Apress, 2012.

[50] S. Ian, "Software engineering eighth edition," 2007.

[51] Y. B. Leau, W. K. Loo, W. Y. Tham, and S. F. Tan, "Software development life cycle agile vs traditional approaches," in *International Conference on Information and Network Technology*, vol. 37, no. 1, 2012, pp. 162–167.

[52] L. Lindstrom and R. Jeffries, "Extreme programming and agile software development methodologies," *Information systems management*, vol. 21, no. 3, pp. 41–52, 2004.

[53] Y. McGivern, *The practice of market research: an introduction.* Pearson Higher Ed, 2013.

[54] E. Mooi and M. Sarstedt, "A concise guide to market research: The process," *Data, and Methods*, 2011.

[55] H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the" stairway to heaven"–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software," in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on.* IEEE, 2012, pp. 392–399.

[56] H. H. Olsson and J. Bosch, "Towards agile and beyond: an empirical account on the challenges involved when advancing software development practices," in *Agile Processes in Software Engineering and Extreme Programming.* Springer, 2014, pp. 327–335.

[57] W. W. Royce, "Managing the development of large software systems," in *proceedings of IEEE WESCON*, vol. 26, no. 8. Los Angeles, 1970, pp. 328–388.

[58] K. Schwaber, *Agile project management with Scrum.* Microsoft Press, 2004.

[59] S. Sharma and B. Coyne, "Devops for dummies," *Limited IBM Edition'book*, 2013.

[60] J. Shore and S. Warden, "The art of agile development. reilly media," *Inc., Sebastopol*, 2008.

[61] V. M. Sue and L. A. Ritter, *Conducting online surveys.* Sage, 2012.

[62] G. Suganya and S. Mary, "Progression towards agility: A comprehensive survey," in *Computing Communication and Networking Technologies (ICCCNT), 2010 International Conference on.* IEEE, 2010, pp. 1–5.

[63] J. Sutherland, "The roots of scrum: How the japanese experience changed global," 1993.

[64] J. Sutherland and K. Schwaber, "The scrum guide," *The Definitive Guide to Scrum: The Rules of the Game*, 2011.

[65] K. YAMADA *et al.*, "Linear models vs agile models: Making the right model decision," *Nagaoka University of Technology CiNii Books*, vol. 2010, no. 50, pp. 85–88, 2010.