Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Web teaching tool for learning agile programming

Bachelor Thesis

2018
Tamara Savkova

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Web teaching tool for learning agile programming

Bachelor Thesis

Program of Study:	Aplikovaná informatika
Field of Study:	2511 Aplikovaná informatika
Department:	Department of Applied Informatics
Supervisor:	Ing. František Gyárfáš, CSc.

Bratislava, 2018
Tamara Savkova

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Tamara Savkova
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: aplikovaná informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Web teaching tool for learning agile programming
Webová výuka agilného programovania

Anotácia: Cieľom bakalárskej práce je navrhnúť a implementovať interaktívne webové prostredie pre výuku vybraných agilných metód programovania. Poskytne používateľovi praktické úlohy pre testmi riadené programovanie, refaktorizačné cvičenia a cvičenia na prácu so zdedeným kódom. Vývojové prostredie pre riešenie cvičení bude webová aplikácia, vytvorený kód spolu s testmi bude zbiehaný vo virtuálnom prostredí na serveri a výsledky testov budú prezentované používateľovi. Aplikácia bude zahŕňať základne nástroje na administráciu systému: administráciu používateľov, správu cvičení a ich výsledkov a verzionovanie zdrojových súborov. Systém bude realizovaný pomocou technológii/nástrojov: JavaScript (jQuery, Bootstrap), MySQL, HTML5, CSS, kompilátory vybraných jazykov, knižnice pre testing, virtuálny server.

Vedúci: Ing. František Gyarfaš, CSc.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 04.10.2018

Dátum schválenia: 15.10.2018 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Abstrakt

Cieľom bakalárskej práce je navrhnúť a implementovať interaktívne webové prostredie pre výuku vybraných agilných metód programovania. Poskytne používateľovi praktické úlohy pre testmi riadené programovanie, refaktorizačné cvičenia a cvičenia na prácu so zdedeným kódom. Vývojové prostredie pre riešenie cvičení bude webová aplikácia, vytvorený kód spolu s testmi bude zbiehaný vo virtuálnom prostredí na serveri a výsledky testov budú prezentované používateľovi. Aplikácia bude zahŕňať základne nástroje na administráciu systému: administráciu používateľov, správu cvičení a ich výsledkov a verzionovanie zdrojových súborov. Systém bude realizovaný pomocou technológii/nástrojov: JavaScript (jQuery, Bootstrap), MySQL, HTML5, CSS, kompilátory vybraných jazykov, knižnice pre testing, virtuálny server.

Kľúčové slová: webová alikacia, testami riadený vývoj, zdenený kód, čistý kód, refaktorizácia, edukačný softvér.

Abstract

The aim of the bachelor thesis is to design and implement an interactive web environment for the teaching of selected agile programming methods. It provides the user with practical tasks for test-driven development, exercises on refactoring and working with legacy code. The web application will be the development environment, the generated code along with the tests will be run in the virtual environment on the server and the test results will be presented to the user. The application will include basic system administration tools: users administration, managing the exercises and their results, and source code version control. The system will be implemented using technologies/tools: JavaScript (jQuery, Bootstrap), MySQL, HTML5, CSS, compilers of selected languages, testing libraries, virtual server.

Keywords: web application, test-driven development, legacy code, clean code, refactoring, educational software.

Contents

Introduction **1**

1 Sources **2**
1.1 Agile software development 2
1.1.1 Extreme programming 3
1.1.2 Methods of XP 4
1.1.3 Practical use of XP 7
1.2 Existing solutions 7
1.2.1 Similar Bachelor and Master theses result works 7
1.2.2 Similar web applications 8
1.2.3 freeCodeCamp 8
1.3 Technologies to be used 9
1.3.1 Client side 9
1.3.2 Server side 10

2 Concept **12**
2.1 Requirements 12
2.1.1 Programming language 12
2.1.2 Courses 12
2.1.3 Development environment 12
2.1.4 Version control 13
2.1.5 Security 13
2.1.6 Error handling 13
2.2 Exercises 14
2.2.1 Exercise types 14
2.3 Architecture 16
2.4 User interface 16
2.4.1 Exercise creation and editing 16
2.4.2 Exercise solving 17
2.4.3 Run output 17
2.4.4 Exercise Toolbar 17

2.5 Data structure 18
2.6 Use cases 18

3 Implementation **19**
3.1 Front-end 19
3.1.1 Angular framework 19
3.2 External modules 20
3.3 Custom modules 21
3.3.1 Editor components 21
3.3.2 Run output component 22
3.3.3 Exercise Toolbar component 22
3.3.4 Use cases 22
3.4 Back-end 22
3.4.1 Virtual server 22
3.4.2 22
3.5 Database 22
3.5.1 22
3.6 Testing module 22
3.6.1 Virtual machine / Container 22
3.6.2 22

4 Sample Course **23**
4.1 Sample course overview 23
4.1.1 Legacy program overview 23
4.1.2 Lesson 1: Debugging a legacy program 24
4.1.3 Lesson 2: Adding new features to the legacy program 24
4.1.4 Lesson 3: Refactoring the legacy program 25

5 Testing **27**

Conslusion **28**

List of Figures

1.1 Planning and feedback loops in extreme programming 3
1.2 Development environment at Codecademy 9

3.1 Quill rich text editor component 20
3.2 Monaco code editor 21

List of Tables

Introduction

Agile Methodology principles were stated by the authors of Agile manifesto, but they were used by software developers years before it happened. Experienced programmers develop own skills while doing mistakes and failures on looking for the best approach of development out of numerous possibilities. Such a skills are sound with agile principles, which, however, seem to be counter intuitive.

that's why it should be learnt...

focus on programming, not projects

Theoretical base is not complete enough for learning agile programming, it requires a of practice. Agile methods have a common feature, they are long-term efficient term Using the agile methods while working on the project and seeing its advantages is a way more convincing.

the aim of the bt...

antifragility?

Chapter 1

Sources

The chapter on sources is devoted to the description of the theory, used for the thesis realisation, overview of the existing solutions of similar problems and introduction of the used technologies.

1.1 Agile software development

The aim of the web application is to introduce and teach how to use the selected methodologies of agile programming. They are based on principles of Agile software development, stated by authors of Manifesto for Agile Software Development [10].

This approach advocates adaptive planning, evolutionary development, empirical knowledge, and continual improvement, and it encourages rapid and flexible response to change [2]. Detailed planning of a large amount of the software process at once for a long span of time is a fragile approach and leads to further growing number of problems, when there is a need of change. The larger the is the system, the more difficult it may be to add new features, and the more difficult is it to fix bugs, and the more resources it requires. The agile methods are rather adaptive than predictive. So they welcome change, they are able to adapt and even thrive on change [17, From Nothing, to Monumental, to Agile].

Agile approach provides this by iterative development, which means a frequent production of working versions of the final system that have a subset of the required features. These working systems are short on functionality, but should otherwise be faithful to the demands of the final system. They should be fully integrated and as carefully tested as a final delivery. The other feature is a feedback mechanism to estimate the situation [17, Controlling an Unpredictable Process - Iterations].

The overview on the methods, on how may they affect the process of software development and how are they represented in the project are covered in the following sections.

1.1.1 Extreme programming

One of the most efficient, time-proven and used frameworks of Agile software development is Extreme programming, also knows as XP. It has already been proven to be very successful at many companies of all different sizes and industries world wide [32].

One of the authors of Manifesto for Agile Software Development American software engineer Kent Beck is a contributor to XP [9]. This method focuses on the practical rather than management aspects of the development. The aim of the XP methodology is to improve software quality and responsiveness to changing customer requirements.

The name "Extreme" is taken from the idea that the beneficial elements of traditional software development are taken to "extreme" levels [9, Preface].

As soon as XP is a framework of Agile web development, it includes iterations and feedbacks. The planning and feedback loops are illustrated on the image 1.1. This approach assumes three levels of plans, the first one for the next a few months, the second one for the next iteration and the last one for the next task within an iteration. The plans are expected to be constantly recreated during the whole development process. Unit testing and acceptance coverage is a very important part of every stage of an XP project. XP provides feedback throughout the whole project on many levels and in many ways [32, Planning/Feedback loops].

Also, while planning, new functionality can be added only when it is necessarily required. Otherwise, most of the unnecessary changed would be removed later, slow down the process and need excessive resources. The design should be simple, so the program would be ready for unexpected changes [31].

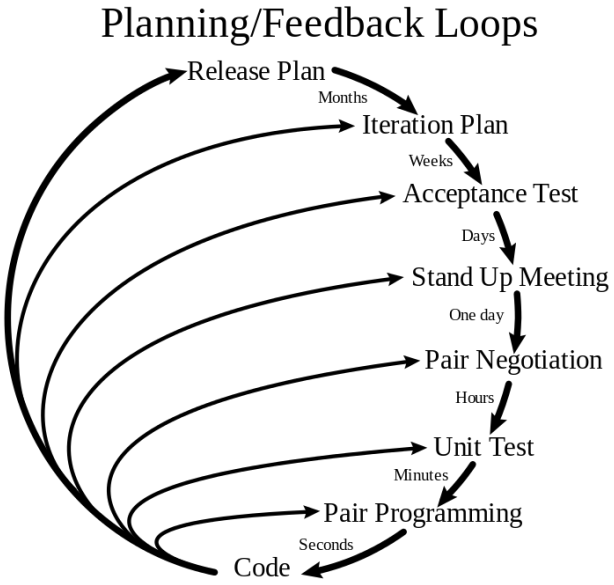

Figure 1.1: Planning and feedback loops in extreme programming

The web application, which would be the result of the work on the bachelor thesis, is designed to encourage the creation of the projects to solve in such a way, so students would develop a program in iterations and would receive a feedback. Each iteration would have exercises, which would lead through the work on the project. A sample project is an example of such a tasks, the exercises a grouped into three iterations. It covers the process of development an adaptive to changes program out of a fragile one, while learning the main XP methods. The sample project is described in detail in Chapter 4.

1.1.2 Methods of XP

XP is built on the best practices, they are combined in a way that they complement and control each other [9, Foreword]. The methods of XP, which were or may be used in the web application, will be introduced in the following subsections.

Test-Driven Development (TDD)

The technique of Test-Driven Development, also known as TDD, was developed by already mentioned Kent Beck, one of the authors of Manifesto for Agile Software Development [8].

This practice supports work in short development cycles. It includes writing of the test cases, which cover the requirements, and only after the implementation is made to pass these tests. Also good written tests provide continuous feedback for the programmer, which is also essential for Agile development [27]. There are different techniques to work with legacy code efficiently, mainly for making modifying it, but TDD is the common feature [15, Test-Driven Development (TDD)].

The key feature of the web application is that it would force using TDD to solve the tasks. A student would program within the provided IDE with editors for the source code and for the tests to be run on it. These tests would be run within the web application and a student would get a feedback. The range of the details provided with test result would be determined by the exercise author.

The sample project shows how work with legacy code can be improved with use of TDD techniques. The first iteration provides exercises on debugging the legacy program. The aim is to find bugs in the legacy program and fix them, but keep the original structure of the code. The user should write his/her own tests to find the bugs and fix them. The second iteration is aimed to add some features to the code, bud to keep the original structure of the code as well. Tests, used in a previous iteration, with the new tests would be needed to accomplish the iteration requirements. The last iteration is made up with numerous exercises on refactoring. The TDD techniques are essential to proof that the program would remain correct.

Refactoring

Code refactoring is the process when a software system is changed in such a way that it does not alter the external behaviour of the code, thus improves its internal structure. The idea of factoring is different from general code cleanup, because it is less risky and invasive. Series of small structural modifications take place. Moreover, the tests provide a confidence that code remains correct [15]. As a result, the code is cleaned up and the chances of introducing bugs reduce. Refactoring is about improving the code, which has already been written. It is important to create a good design firstly and then to code. But refactoring is the opposite of this practice, because it deals with bad, chaotic design to rework it into a good one [18, Preface].

Refactoring is a common part of an efficient work with legacy code, when there is a need in the design improvement. Such a software change would make its structure more maintainable, while the behaviour would be kept [15].

Also there are cases, when refactoring is not recommended yet. For example, when there is a need to rewrite the whole code from the beginning instead. It may happen, when the code works mostly incorrectly. It is also recommended to avoid refactoring, when the deadline is close, because refactoring may remain unfinished. It would lead to a raise in cost of maintenance as well as an code, which was not reafactored at all [14, Preface].

The web application also would make practice with refactoring possible. Refatoring a legacy code exercise are introduced in the sample project. Since refactoring should not be made when there are many mistakes in the code, the refactoring exercises take place in the third, last iteration. At this moment, after the previous two iterations are completed successfully and passed all the tests, the code would be correct. Each of the numerous exercises is about a single step of the code change, exercise description and title would reference to the book "Clean Code" by Robert C. Martin [23]. It is one of the mostly recommended books for software development, and for a good reason. During and after refactoring the code should maintain correct, and it can be provided by tests, written by the user.

Mature optimisation

Optimisation is similar to refactoring, they both keep the functionality of the code. Although they have different goals. Refactoring changes the program structure, so it is easier to maintain. Optimisation makes changes in the resources, used by the program, e.g. time or memory [15]. Refactoring makes code readable and understandable, while optimisation of the code to achieve the needed performance often makes code harder to understand [18].

Optimisation may not always be performed optimally and save the resources as

a result. Donald Knuth, an American computer scientist and professor at Standford University, mentioned this issue in his paper "Structured Programming With go to Statements" [21]. He wrote that the programmers waste too much time on worrying about the speed of not crucial software parts, what makes negative impact, when maintenance and debugging are considered. So he concludes, that premature optimisation is the root if the all evil. As soon as optimisation may reduce readability and make the system more complex, it may be harder to maintain and debug it. Because of that, it is realised at the end of the development. This practice is sound with Agile development and XP principle of development of only the necessary parts only when they are needed.

The web application also enables an illustration of this point to a student. The sample project contains an exercise on refactoring, where it is required to change the way of computing the result. Such a task is within the last steps of the refactoring process. The steps after make no influence on the structure.

Pair programming

Pair programming is a technique, when the code for a single task is written by two people. The first programmer, the driver, has control of the computer and actively implements the program, and the other, the observer, provides feedback with reviewing the written code and thinks strategically about the work direction [33]. Feedback is also provided during pair programming.

Web application would allow to work on assignments in pairs.

User story

User story is a customer's functional requirement.

Ron Jeffries, one of the authors of the Manifesto for Agile Software Development [10], proposed a "Three Cs" formula for user story creation. The first "C" is for Card, on which user stories are written. It does not contain all the information that makes it up, but it has the needed information to identify it. There may be notes of cost and priority of the requirement. The second "C" is for the Conversation between the stakeholders, which particularly takes place during the planning. The third one is for the Confirmation, which is provided by an acceptance test, telling is the aims of the conversation have been reached [19].

In XP it may be used as an instrument for stating the requirements and providing a feedback, also the tests may be based on user stories. In the web application the user stories would be used to formulate the requirements needed to fulfil to accomplish an exercise.

1.1.3 Practical use of XP

According to the 12^{th} annual State of Agile survey, which was taken in 2018, only 1% of respondents' organisations use XP [3]. Another Agile framework, Scrum, is used by the majority of 56%. However, hybrid of another Agile framework, Scrum, and XP, is used by 6% of companies. This hybrid, known as ScrumXP, combines practices of Scrum project management and programming practices of XP.

Extreme programming is not that popular practice because of several reasons. The following reasons are the ones which are associated with the mentioned techniques.

It may seem irrational, even impossible to write tests before the code exists, which is the idea of Test-Driven Development. Even writing tests in general is not a common practice. As mentioned previously in a corresponding subsection, the benefits are real.

As about pair programming, it is counter intuitive as well. But as a result, the software quality may be increased within a lesser time than if two programmers worked separately. On one hand, programmer working by himself gets distracted more often, pair programming simplifies testing, getting feedback and code review, programmers feel more confidence in their. But on the other hand, it cannot work for every programmer or company. Most of the developers, especially the experts, prefer working on their own [13, Two by Two].

The aim of the web application is to explain such an approach, show examples of use and to teach how to get benefits from it.

1.2 Existing solutions

The web application would be presented in a form of an interactive online course. Its interface and control would be inspired by the existing platforms, introduced in the following subsections.

1.2.1 Similar Bachelor and Master theses result works

Results of work on Bachelor and Master theses at Comenius University in Bratislava, faculty of Mathematics, Physics and Informatics will be introduced in the following subsections.

Web learning programming using TDD technology

This Bachelor thesis (original title: Webová výuka programovania pomocou metodológie TDD) was written by Tomáš Bočinec in 2018.

The resulting web application is a good example of a simple and functional learning environment. The web application for learning Agile programming methods would use

similar mechanism to compile the provided source code and tests, to run the tests and return the result. But more modern and efficient technologies would be used. And the exercises would have more specific structure for practising the Agile programming methods. The details are introduced in the next section 1.3 on used technologies.

1.2.2 Similar web applications

Codecademy

Codecademy is an education company, which positions itself as an alternative to classical education. Web application they provide is a result of rethinking current approach on education [12, About].

There are courses of Web Development, Programming and Data Science. In fact the exercises focus on programming languages syntax, the knowledge of which depends mostly on practising. Basic principles behind it, important for the beginners as well, are left without the attention. These principles are mentioned and learned a lot at universities, which are criticised by Codecademy. However, the courses are great for the ones, whose aim is to gain knowledge of a programming language itself without any learning material above.

Codecademy popularises Test-Driven Development, and offers a course on it for an enrolment payment. Also the web offers code reviewing and receiving a feedback, but Codecademy does not provide this functionality directly and the other service Git Hub is used for it.

The web application for learning Agile programming methods would introduce the idea behind the techniques used in exercises, otherwise they would remain to seem counter intuitive and not understood.

Codecademy web page and development environment are very user-friendly and pleasing to the eye. The interface of the web application for learning Agile programming methods would be designed to produce the same impression. It is shown on the screenshot: 1.2. The environment is divided into three panes. The left contains the exercise content, the middle one is an editor, and the right one shows an output in the built-in web browser.

1.2.3 freeCodeCamp

Another online courses interactive platform is freeCodeCamp. Moreover, it is a community, which helps to learn programming for free and to get experience by contributing to open source projects used by nonprofit organisations. They offer practising with completing coding challenges and work on the projects. Also joining the study group in the students' city to code in-person is encouraged. On the opposite to Codecademy,

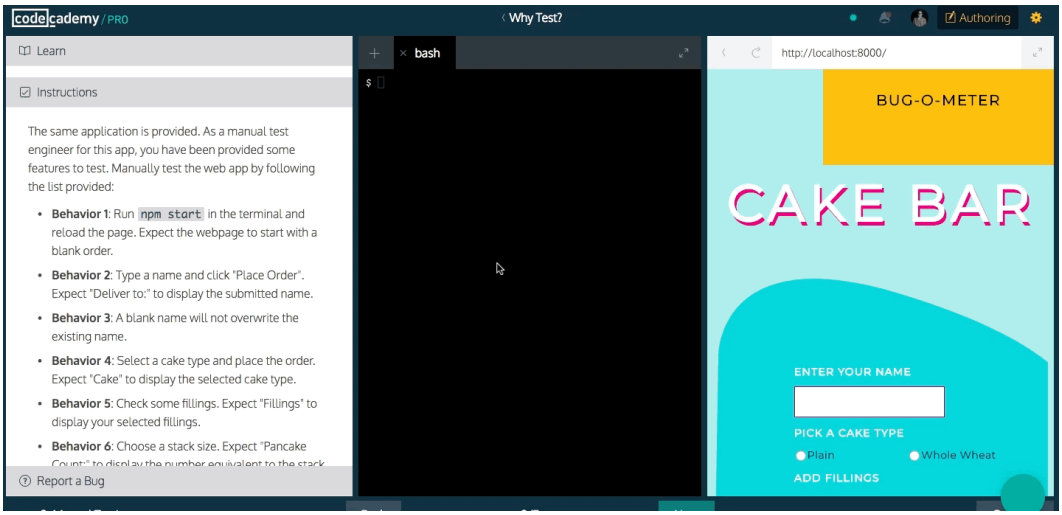

Figure 1.2: Development environment at Codecademy

freeCodeCamp does not position itself as a high education alternative. They encourage to study to earn a degree and to pursue the courses they provide [16].

Some of the courses take form of projects, this approach would be used in the web application for learning Agile programming methods. The reason is that it depicts the principles and enables practising in the best way.

Like in Codecademy, the web does not provide a functionality of code review of the projects directly. The other service Git Hub is used for it. They introduce a term User Story while working at the projects to introduce the project requirements. In the web application for learning Agile programming methods the user stories would be used to formulate the requirements needed to fulfil to accomplish an exercise.

1.3 Technologies to be used

The technologies, which would be used in the web application for learning Agile programming methods, are introduced and analysed in the following subsections. These technologies are used in the web application prototype.

Web application would use client-server architecture. The server part would provide the central functionality. The clients would connect to it and request performing the tasks, and the server would provide the results.

1.3.1 Client side

The client side application would use Angular. It a structural framework for dynamic web applications, which is based on JavaScript pment to initialize, develop, scaffold, and maintain the application, add new components, modules and services [5]. Angular enables development of modular and reusable dynamical web applications.

TypeScript

The Angular framework is built entirely in TypeScript, and it is used as a primary language. As soon as browsers cannot execute TypeScript directly, the code must be converted into JavaScript with tsc compiler [4, TypeScript Configuration].

HttpClient

The front-end of the application communicates with backend services over the HTTP protocol. The web browsers support APIs for making HTTP requests of two different types: the XMLHttpRequest interface and the fetch() API. Angular offers the HttpClient, which is a simplifies HTTP API that rests on the XMLHTTPRequest interface exposed by browsers. Also it includes testability features, typed request and response objects, request and response interception, Observable APIs, and streamlined error handling [4, HttpClient]. Angular makes use of observables as an interface to handle a variety of common asynchronous operations. The observables provide support for passing messages between publishers and subscribers [4, Observables & RxJS].

Sass

User interface would be implemented using HTML5 and CSS3. Sass, a preprocessor scripting language, would be compiled into CSS. The CSS preproccessors are responsible for recompiling the existing code into the format compliant to the CSS standard. Sass provides such a functionality as multilevel nesting, what improves readability of the code and reduces its volume. It also allows to declare variables. CSS3 also has such a feature, but it is not supported by all the browsers, i.e. Internet Explorer. Mixin insertion solves the problem of a need to duplicate the code. Also Sass enables expanding, inheriting and importing the styles [28].

1.3.2 Server side

The server side Java application would use Spring Framework 5.0. The core functionality of the web application is compilation of the source code and tests, running these tests and returning the result. When the server part would get a request to provide this functionality, the code would be compiled.

Maven

Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Maven automates such a tasks as downloading dependencies, putting additional jar files on a project classpath, compiling

source code, running tests, packaging compiled code into deployable artifacts such as JAR, WAR and ZIP files, and deploying these artifacts to an application server [24]. Such an automatising assures avoiding the mistakes, which may occur when building the software manually.

Spring

The Spring Framework is an open source developed by Pivotal Software application framework and inversion of control container for the Java platform. It provides comprehensive infrastructure support for developing Java applications.

It contains modules, which make development faster and simpler. Spring Boot is an extension of the Spring Framework, a convention-over-configuration solution. In other words, it makes it easy to create applications, which can "just run" [29].

Database technologies

Database would be used to store data about users, their activity, created and submitted assignments. Spring Data would be used to provide Spring-based programming model to access the data and relational database PostreSQL. Spring Data JPA module would be used to implement JPA based repositories [30]. JPA is a Java Persistent API, which provides an object/relational mapping facility for managing relational data in Java applications [20, Part VI, Persistence].

Testing technologies

Java source code and tests would be used within the web application. The server side application would receive a request to compile the provided source code and tests. JUnit 4 framework would be used to write the tests, which is very convenient and intuitive in use.

There would be two types of the tests. The tests of the first type would be written by the students as a part of exercises. The tests of the second type are written by the exercise author, and they are not accessible by student from the web application. The exercise author would also set the scope of the viewed test results. It ranges from providing of the whole information about the tests expected and actual values, to the providing of the information if the test was passed only.

The compilation and running the code would take place on a virtual machine for security reasons.

Chapter 2

Concept

This chapter is devoted to the specifications and concept of the resulting project, which is a web application. The application would enable to create, manage learning materials and solve them in an interactive way. The materials author would be further called *teacher* and the solver would be called *student*.

2.1 Requirements

This section is devoted to the requirements for the resulting web application and their purpose.

2.1.1 Programming language

The web application should support use of Java programming language of versions 8 to 10. Testing framework JUnit 4 would be used for testing. The application should support compilation of Java programs and running the tests.

2.1.2 Courses

The materials would have a structure of courses. Each course would contain lessons, groups of exercises. Application would provide tools for creating the courses, lessons and exercises, edit them and delete. Courses and lessons would have title and description, and it would be able to reorder them. There would be five types of exercises, they are specified in detail in section 2.2.1 Exercise types.

2.1.3 Development environment

The web application would have convenient development environment, and it would be close to conventional IDEs. It would consist of code editors with multiple tabs, these tabs should be easy to add and remove. The environment would enable code execution

and providing a feedback. As soon as the programming language is Java, the feedback should include possible compilation errors and tests results.

2.1.4 Version control

Another requirement for the resulting application is version control of the student's solutions. The web application should store and show source code, tests and files, which are submitted and run by student.

When solving exercise, student would be able to view previous versions and insert their files to the editor. He would be able to access the files from any exercise within a current course. As soon as there may be a very large amount of solutions, they all would not be loaded at once, but user would be able to load them page by page. Solutions would be ordered by date and time of submitting and user would be able to filter them by completeness.

This feature is useful for refactoring or working with legacy code, e.g. when student needs to extend tests from previous solution with new ones.

2.1.5 Security

Another important requirement concerns about security issues and elimination of the potential risks.

Running the programs, written by teachers and students, on the server brings risks for the system. Running the program would be isolated, and its would have limited access to the rest of the system. It may be aimed with use of a form of a software vitalisation, a *virtual sandbox*.

Another web application security risk is *Cross-Site Scripting* (XSS), which allows malicious code to be added to a web page via user input, e.g. form submissions, and used in it [26, A7-Cross-Site Scripting (XSS)]. To prevent this, the input would be sanitised: rather than accept or reject input, another option is to change the user input into an acceptable format [25, Sanitize]. Such an attack may be used in text editor for an exercise description.

2.1.6 Error handling

Web application should be robust, what can be achieved with efficient approach for errors handling. Such an approach should provide informative error messages and good user experience.

2.2 Exercises

Practising agile programming methods would be realised by solving exercises. To practice various agile programming methods, it is not enough to have an environment, which supports only work with source code and estimates solutions automatically. All of these methods include unit testing, so works with tests would be included. Support of the files would expand the possibilities of testing.

2.2.1 Exercise types

There would be five types of exercises to provide different ways to practice agile programming methods separately or in combinations. Their concepts, purpose and usage would be described in the following sections.

Theory

Exercise type *Theory* is the basic one, it is aimed to provide information. May be used mainly to introduce or conclude a current course or lesson, or to make an overview on legacy code.

It would provide only a text area and would be marked as solved at the moment, when is viewed by the student. On exercise creation and update, the teacher would be able to use text editor with ability to format, e.g. to provide example source code. The formatted text would be provided to the student.

Interactive Exercise

Interactive Exercise type would provide tools for programming. On exercise creation, the teacher would be required to fill title and description of exercise. Then he would use editors to bring in source code, initially provided to the student. It would show basic code structure to the student, but also may be useful for further work with legacy code. Next editors would be designed for tests. Teacher would use them to fill hidden tests to control the solving progress. Also he would provide tests, which would be initially provided to the student, e.g. to recall the JUnit 4 syntax. Teacher would be able to simply duplicate the hidden version to fill the shown one, but also to create two different.

On students side, it would contain an exercise description and two editors, making a development environment. The first one is for source code, and the second one is for tests. The student would use the environment to solve an exercise, and he would be able to run his code withing the web application and get feedback. The provided feedback would be consisted of two parts. The first one would tell about compilation

and tests results for the students code from the development environment. The second one would be based on the teachers code, which would not be available for the student.

Interactive Exercise with Files

This exercise type provides the same features as previous one, and expands its tools working with files. One more text editor would contain files, and there would be teachers and students versions as well. These files would be used for the purpose of testing.

Black Box

The next exercise type is named *Black Box*, what means a system or a function, for which only its input and output is known. This term describes the main feature of this type, since the student would not see source code, but would send input and get some form of output from the program. This tool is useful for practising unit testing and test-driven development. On working on this exercise, the student would try to find mistakes in the hidden program with the use of tests.

To create an exercise of this type, teacher would fill its title and description. The latter should contain code structure or functions names, or any kind of information to give the student knowledge about the hidden code. So he would be able to call necessary functions from his tests. Except filling an exercise title and description on exercise creation, the teacher would enter controlling source code, which would be hidden from the student, would be compilable and contain several logical mistakes. The application would provide information for the teacher on how to design the source code, so it would be used correctly. The tests, which would be shown to the student, would be filled as well. The purpose of these hidden tests is the same as for such a tests in exercise type *Interactive Exercise*.

The student would write tests, which would not be passed be the program. The student would run his tests for the hidden program, and would get feedback, telling the number of found mistakes and their total number.

The application would provide this functionality and solving *Black Box* exercise is a correct way. Incorrect solutions would be ignored and not counted to the solution. Examples of such a solutions are the following: the student may write test, which would not be passed, but it would not find the mistake. It may happen, if this test would not pass a correct version of the hidden program as well. This solution should be ignored. Another example is writing several tests, so they would find the same mistake several times. Only one test would be counted for solution, the rest would be ignored.

Black Box with Files

This exercise type extends previous one with ability to work with files, which would be used for testing.

2.3 Architecture

...

2.4 User interface

This chapter is devoted to user interface components concept.

2.4.1 Exercise creation and editing

Exercise creation and editing would have the same structure, with difference in handing the user input and availability of initially defined values. The input fields would change dynamically, depending on the exercise type of the exercise. It would be possible to change exercise type without losing data the fields contain.

Non-dynamic part would contain fields for exercise title, description and exercise type selector. When an exercise type is chosen, the form would change and the corresponding editors appear. Every editor component would have the same structure, and differ only with content and handling the user input.

There would be three types of editor. The first one would be a source code editor, used for exercises of type *Interactive Exercise* and *Interactive Exercise with Files*. The second would be tests editor and used for exercises of type *Black Box* and *Black Box with Files*. And the last one would be text editor, used for exercises of type *Interactive Exercise with Files* and *Black Box with Files*.

The first type *Theory* would contain only description of the exercise. Exercise type *Interactive Exercise* would have editors for source code and tests, and exercise type *Interactive Exercise with Files* would have one more editor for text files.

One of the required features, stated in the corresponding section 2.1 Requirements, was convenient creation of two versions of source code, tests and text files. The shown version is for testing and would be available only for a teacher, and the hidden version is for showing to a student when solving. On exercise creation or update, there would be an editor for a hidden version, and there would be a select component below. The component contains two options for the shown version. The first option, “the same“, would be used when an exercise author wants the shown version to be the same a hidden. And the second option, “custom“, would be used to create or edit a different

shown version, and an new editor element appears. Some of the exercise types would require only a hidden or shown version of source code or test, so there would be no select component.

The submit button is enabled as soon as the form input is valid. On submit, there would appear a notification telling if the data were saved successfully. When cheating new exercise, there would be an suggestion to create another one.

2.4.2 Exercise solving

When solving an exercise, its view is determined by its type. Every exercise types has view and functionality according to the concept, described in a section 2.2.1 Exercise types.

Every exercise would contain a component for displaying a description. The description content would be created with use of rich text editor. Editor components from the previous section are used for exercise solving as well.

Every editor component would be extended with version control component, as it required at the concept section 2.1.4 Version control. It would enable resetting the editor content and loading content from a submitted version.

2.4.3 Run output

On solving exercise of any type, except *Thesis*, there would be available a "Run" button for estimation a current result. When an exercise is submitted for the first time, an output area would appear. During the estimation process, it would contain "Loading..' message. When feedback would be ready, it is would be printed out within the output. Feedback would contain compilation and testing result for both public and private versions, and represented as percentage solving progress.

Estimation would run two times for exercise types *Interactive Exercise* and *Interactive Exercise with Files*. First time for public version (running student's tests on his sources) and second time for private version (running teacher's tests on student's sources). The progress would be represented as percentage and mean how much tests were passed out of their total number.

Testing result for exercise types *Black Box* and *Black Box with Files* would contain only number of revealed mistakes and total number of them. The progress would be represented as percentage, corresponding to the found mistakes number.

2.4.4 Exercise Toolbar

The pages for solving the exercises would contain two equal toolbars, one on the top and second on the bottom of the page. They would contain the mentioned "Back"

button, navigation within the lesson exercises, and there would be a flag representing the progress on the solving current exercise. Toolbar view would change dynamically on an exercise change.

2.5 Data structure

...

2.6 Use cases

...

The main page provides a list of the created courses and information on them. The user may choose one of the listed courses to enter it. Also the user may manage the courses, i.e. create, edit or delete them, with the use of the button. This and the rest of the pages contain “Back“ button for easier navigation.

Each course page contains an information about it. There is an expansion panel below with lessons and exercises they include. Every lesson and exercise may managed as well. Management includes lesson editing, deleting, reordering and creation.

...

Chapter 3

Implementation

This chapter is devoted to the implementation of the web application, which is the result of the bachelor thesis. It was realised according to the concept introduced in the previous chapter 2 Concept to meet the stated requirements.

This application uses full-stack web framework as soon as it supports front-end interface, back-end services and databases development. The application uses client-server model, where front-end corresponds to the client side, and front-end is considered as server side. The following sections describe implementation of every technologies stack component.

3.1 Front-end

...

3.1.1 Angular framework

Front-end uses Angular, a platform and framework for building dynamic applications on the client side. Its primary programming language is TypeScript. Like any Angular application, the result application depends on features and functionality provided by libraries that are available as npm packages, and npm package manager was used to download and install them [1, Node.js].

Angular modules

Angular provides modules, which provide components implementing common interaction patterns according to the Material Design specification [6, Components]. Components such as Toolbar, Table, Paginator, Dialog, Expansion Panel etc. were used for creating fast, modern and consistent user interface.

Angular Reactive Forms module

Angular Reactive Forms module was used to create complex dynamic and nested forms. They allow storing inputs from the user or generate multiple nested forms with the same structure. They were used to create and edit courses, lessons and exercises, and also to solve the latter. As a result, it is possible to change the form view as soon as different exercise type is selected. Also they are used to add, remove and rename files at the code editors on exercise creation.

Accuracy and completeness of the input is improved with the form validation, provided by the Form Group. The required fields change dynamically with the form itself.

3.2 External modules

Except Angular modules, application uses two external modules, which provide editors. Functionality and use of them is described in the following sections.

Quill module

The first one is based on Quill rich text editor. It was chosen because its component provides useful text formatting features, i.e. source code, quotations and headers. There is a view of a used quill editor at screenshot 3.1. This component is used to create and edit exercises description.

The code editor is configured to store user input as HTML-string. The editor uses Angular DomSamitizer to sanitise these values, so it also meets a requirement of security.

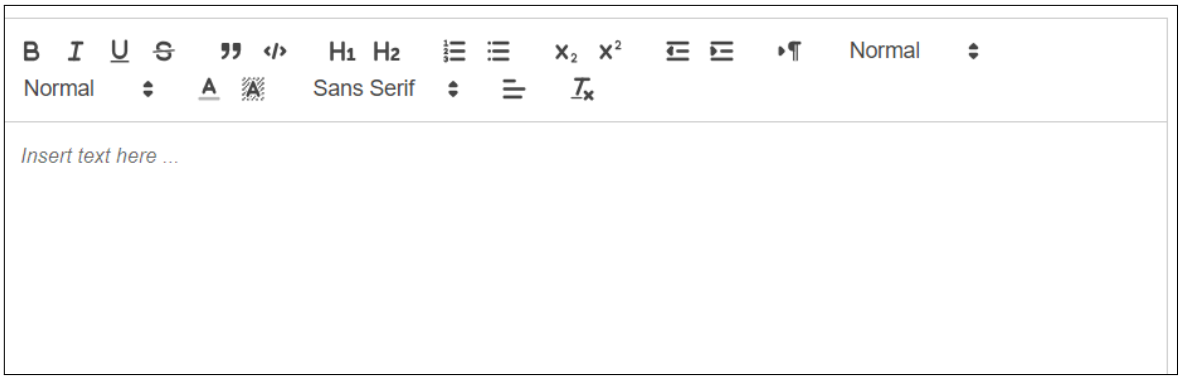

Figure 3.1: Quill rich text editor component

Monaco module

The second module is based on Monaco code editor, which powers a high productivity Visual Studio Code editor. Its component was chosen because it has numerous tools

to create a powerful development environment. The editor enables highlighting of the Java code, searching, suggestions, minimap, so the requirement of providing a close to conventional IDE was met. There is an example of use of Monaco editor module at screenshot 3.2.

Figure 3.2: Monaco code editor

3.3 Custom modules

Angular provides tools for making program modular. There are three modules in the application: courses, lessons and exercises. They contain components, routing modules, service providers. Components of these modules present data and define the view together with HTML templates. Routing modules enable navigation from one view to the next as users perform application tasks. The service providers handle sending requests to the server and retrieving the responses [1, Architecture].

They are used to display pages or sections. Components are reusable, so some of them are used for different purposes, e.g. to create and edit a course, to manage a course and a lesson. Also they are used to display repetitive sections to avoid duplicates, e.g. show three same code editors with different content.

3.3.1 Editor components

Most of the use cases include some kind of user input. There are used different types of editors, from simple input fields to powerful code editors. Their implementation would be described in the following chapters.

Exercise creation and editing

Exercise creating and editing is implemented with the same component. Angular Reactive Forms were used to interact with the user and manage his input.

— It would be possible to change exercise type without losing data —

Exercise solving

3.3.2 Run output component

3.3.3 Exercise Toolbar component

3.3.4 Use cases

3.4 Back-end

3.4.1 Virtual server

3.4.2 ...

3.5 Database

3.5.1 ...

3.6 Testing module

3.6.1 Virtual machine / Container

3.6.2 ...

Chapter 4

Sample Course

In order to present the functionality and usage of web application, a sample course was created. Its exercises include all the exercises types and agile programming methods. The following section will lead through the content of the course.

4.1 Sample course overview

There are three lessons in the course, each represents an iteration of the work on the program. Every lesson contains exercises, which lead through the work on the project. The course is designed in such a way, so the user would learn agile programming methods and apply his skills while working on the project. Most of the exercises would cover multiple skills, like test-driven development, unit testing, refactoring and working with legacy code.

Section *Legacy program overview* gives more information about the program, which is worked on during the course. Further sections *Lesson 1 to 3* describes lessons content, purpose, and which agile programming methodologies do they introduce.

4.1.1 Legacy program overview

The student would work on an interactive Reversi game, based on a legacy program. It is a console application for two players. Each Reversi piece has a black side and a white side. The state of the game is read from configuration files, which have the following structure: first row has "B" (black) or "W" (white), which means the player on turn; the second row contains positions of black pieces; and the third row contains positions of white pieces. When the game starts, program asks a player on turn to enter tile position to move on. The game ends when player on turn has no tiles to move on. The one, who has more tiles on the board, wins the game.

4.1.2 Lesson 1: Debugging a legacy program

The first lesson is called *Debugging a legacy program* and it contains three exercises, which step by step lead through looking for mistakes and fixing them.

As soon as the programmer does not know how legacy program works, even if he is the author, it becomes more difficult to fix mistakes and not produce more. And using tests becomes more crucial than usually. The purpose of the following exercises is to show how testing is useful for fixing program correctly and efficiently.

Exercise 1: Intro

The first exercise is introductory. It contains no assignments, but acquaints the user with general information about the course, its lessons, exercises and the legacy program.

Exercise 2: Finding the Bugs in Legacy Program

Work with legacy code starts from the second exercise. The user would get program, which contains three mistakes. As soon as this exercise is of type *Black Box with Files*, the user would be provided only with development editors for tests and files. He would not be able to see or change source code of the program.

Exercise is designed in such a way, so the user would practice test-driven development. The user would need to write tests to find the mistakes. On submit, the user would get a feedback, telling how much errors did he reveal and their total number.

The mistakes relate to marginal cases and incorrect user inputs. Original program works incorrectly on such an inputs: move on tile, which is not empty; move on not adjacent tile, what violates the rules; and move out of the bounds of the board.

Exercise 3: Debugging the Legacy Program

When user reveals mistakes in the previous exercise, these mistakes would be corrected in this one. It is recommended to use tests, which were submitted in the previous exercise, and web applications provides such a feature to make work with legacy code more convenient. This exercise is interactive, so user would solve it with changing the program files, running the modified program and get feedback on his solution.

4.1.3 Lesson 2: Adding new features to the legacy program

The second lesson is devoted to the most often kind of work with legacy code, which is adding a new feature. Adding new functionality may be as risky as fixing the mistakes, so proper testing is important too. The aim of this lesson is to show how unit testing can be helpful for making sure, that program remains functional. Also this lesson would introduce of safe and efficient methods of changing legacy code.

Exercise 1: Board of Any Size

Exercise 2: Show Hints

4.1.4 Lesson 3: Refactoring the legacy program

The last lesson is devoted to the most efficient way to tidy code and make it easier to aim three factors of code quality and cleanliness: readability, maintainability and extensibility. Refactoring should be done in the end and no other code changes should be done at the same time. Because of this, this iteration of programming is done within the last lesson of the course. Each of the following exercises is devoted to one of the techniques of refactoring and leads step by step through them. These techniques will be practices on the legacy code from previous lessons, as soon it is a good example of program with lack of refactoring, and it becomes a good proof of necessity of it. Exercise titles refer to related chapters of the book *'Clean Code: A Handbook of Agile Software Craftsmanship'* by Robert C. Martin. Also exercises contain citations from it, which describe the reason and idea behind the used techniques in the best way.

Exercise 1: Constants versus Enums

This step of refactoring is aimed to make the code more readable. Original legacy code uses integers to represent cells to tell if a black, white or no piece is placed on it. The task is to replace these repetitive numbers with constant variables. In this case it is more reasonable to prefer *enum* with name *Player* rather than *constant*, as soon as its meaning cannon be lost, because they belong to an enumeration that is named [23, Constants versus Enums].

Exercise 2: Do One Thing

The second exercise introduces an important principle of programming, which is applied to programming entities on many levels, e.g. lines of code, functions, classes, and even refactoring itself. The legacy code contains several cases of violating this principle[23, Do One Thing].

The aim of the exercise is to extract new functions from the functions, which do more then 'one thing'. As a result, work with code becomes more efficient. It becomes possible to test application on more levels, to use methods of changing legacy code is needed, and to understand what function does at one glance.

Exercise 3: No Duplication

The next step of refactoring is described as '*the primary enemy of a well-designed system*' by Robert C. Martin, and he explains why: "*It represents additional work,*

additional risk, and additional unnecessary complexity[23, No Duplication].“

Source code from the exercise contains repetitive code, and the task is to eliminate such a duplicitous with extracting new functions and use them. The most noticeable advantage of this change is that instead making changes in every occurrence of such a code, it is enough to change it once in one place.

Exercise 4: One Level of Abstraction per Function

When the core steps of refactoring are completed, this exercise introduces often neglected step of refactoring, which improves code readability significantly.

This exercise suggests the same solution of extracting methods as the previous ones. As a result of this separation, it becomes clear which lines of code express an essential concept or a detail [23, One Level of Abstraction per Function].

Exercise 5: Small!

When a function is too big to fit into the screen, it makes it clear, that it should be refactored [23, Small!]. Previous refactoring steps lead not only to the mentioned benefits, but also reduce size of the functions. Most of them used extraction of one functions from another to aim the goals, but this exercise suggests one more approach.

The task is to change the structure of storing values, which represent the number of black and white pieces on the board. When using a map instead of two integers, one of the function significantly reduces, and readability is not only maintained, but also improved.

Exercise 6: Error Handling I & Exercise 7: Error Handling II

The last two exercises conclude the lesson with leading through handling the errors. They belong to the lesson on refactoring, as soon as it is important to write clean code to handle errors.

There are two kinds of errors, so work on them is separated into two exercises. Legacy code uses a harmful practice of printing messages to console and returning *void*, when any error occurs. It leads to unexpected program behaviour, which may be difficult to reveal, repeat and correct. The exercises suggest use of custom exceptions to catch errors efficiently.

Chapter 5

Testing

...

Conclusion

The main aim of the bachelor thesis was to develop the web application, which would introduce and provide environment for practising agile methods of programming. As soon as these methods are initially taken as counter intuitive, it is even more important to try them to understand why they are considered to be efficient and useful. An interactive form enables leading through every step, explaining it and providing feedback.

...

Bibliography

[1] Angular: Getting started, note=`https://angular.io/guide/quickstart`, last accessed on 06/05/19, key=angular$_g$*etting*$_s$*tarted*.

[2] Agile Alliance. What is Agile Software Development?, 2013. `https://www.agilealliance.org/agile101/`, last accessed on 06/05/19.

[3] The 12th annual State of Agile Report. `https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report`, last accessed on 06/05/19.

[4] *Angular Guide*. `https://angular.io/guide/`, last accessed on 06/05/19.

[5] *Angular Guide*. `https://angular.io/cli`, last accessed on 06/05/19.

[6] Angular Material: Components. `https://material.angular.io/components`, last accessed on 06/05/19.

[7] Angular: Reactive Forms. `https://angular.io/guide/reactive-forms`, last accessed on 06/05/19.

[8] Kent Beck. *Test Driven Development: By Example*. O'Reilly Media, 2002.

[9] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. O'Reilly Media, 2004.

[10] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development. `http://agilemanifesto.org/`, last accessed on 06/05/19, 2001.

[11] Tomáš Bočinec. Webová výuka programovania pomocou metodológie TDD, 2018.

[12] Codecademy web page: About. `https://www.codecademy.com/about`, last accessed on 06/05/19.

[13] Lee Copeland. Extreme Programming. *Computerworld*, 2001. `https://www.computerworld.com/article/2585634/app-development/extreme-programming.html`, last accessed on 06/05/19.

[14] Ward Cunningham. Ward Explains Debt Metaphor. `http://wiki.c2.com/?WardExplainsDebtMetaphor`, last accessed on 06/05/19, 1992.

[15] Michael C. Feathers. *Working Effectively with Legacy Code*. Prentice Hall PTR, 2005.

[16] freeCodeCamp web page: About. `https://about.freecodecamp.org/`, last accessed on 06/05/19.

[17] Martin Fowler. The New Methodology. *Wuhan University Journal of Natural Sciences*, 2005. `https://www.researchgate.net/publication/225478666_The_New_Methodology`, last accessed on 06/05/19.

[18] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Reading, MA : Addison-Wesley, 1999.

[19] Ron Jeffries. Essential XP: Card, Conversation, Confirmation. `http://ronjeffries.com/xprog/articles/expcardconversationconfirmation/`, last accessed on 06/05/19.

[20] *The Java EE 6 Tutorial*. `https://docs.oracle.com/javaee/6/tutorial/doc/bnbpy.html`, last accessed on 06/05/19.

[21] Donald E. Knuth. Structured Programming with go to Statements. *ACM Computing Surveys (CSUR), last accessed on 06/05/19*, 1974. `https://dl.acm.org/citation.cfm?id=356640`.

[22] Daniel Krajč. Webové vývojové prostredie pre jazyk C++. Master's thesis, Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky. Katedra aplikovanej informatiky, 2010.

[23] Robert C. Martin. *Clean Code. A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, 2009.

[24] *Apache Maven Project: What is Maven?* `https://maven.apache.org/what-is-maven.html`, last accessed on 06/05/19.

[25] Data Validation: Sanitize. `https://www.owasp.org/index.php/Data_Validation#Sanitize`, last accessed on 06/05/19.

[26] Top 10-2017 A7-Cross-Site Scripting (XSS), 2017. `https://www.owasp.org/index.php/Top_10-2017_A7-Cross-Site_Scripting_(XSS)`, last accessed on 06/05/19.

[27] David Parsons, Ramesh Lal, and Manfred Lange. Test Driven Development: Advancing Knowledge by Conjecture and Confirmation. *Future Internet*, 2011. `https://www.researchgate.net/publication/220103002_Test_Driven_Development_Advancing_Knowledge_by_Conjecture_and_Confirmation`, last accessed on 06/05/19.

[28] *Sass basics*. `https://sass-lang.com/guide`, last accessed on 06/05/19.

[29] *Spring Boot: Overview*. `http://spring.io/projects/spring-boot`, last accessed on 06/05/19.

[30] *Spring Data: Overview*. `http://spring.io/projects/spring-data`, last accessed on 06/05/19.

[31] Don Wells. Never Add Functionality Early. `http://www.extremeprogramming.org/rules/early.html`, last accessed on 06/05/19, 1999.

[32] Don C. Wells. Extreme Programming: A Gentle Introduction. `https://www.researchgate.net/publication/246155995_Extreme_Programming_A_Gentle_Introduction`, last accessed on 06/05/19, 2003.

[33] Laurie Williams. Integrating Pair Programming into a Software Development Process. *CSEET '01 Proceedings of the 14th Conference on Software Engineering Education and Training*, 2001. `https://dl.acm.org/citation.cfm?id=794899`, last accessed on 06/05/19.