Play with Scala

Functional Programming Course

Martin Zuber, B00066378

Project Supervisor: Luke Raeside

January 27, 2016

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Degree of Honours B.Sc. in Computer Science in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Acknowledgements

In performing my assignment, I had to take the help and guideline of some respected persons, who deserve my greatest gratitude. The completion of this assignment gives me much Pleasure. I would like to show my gratitude to the project supervisor Dr Luke Raeside, Institute of Technology, Blanchardstown for giving me good guidelines for the thesis throughout numerous consultations. I would also like to expand my deepest gratitude to all those who have directly and indirectly guided me in writing this thesis, especially Dr Markus Hofmann.

Abstract

This paper is dedicated to computer programming with the focus on programming styles found in modern application development. The idea for the project came to life when I, the author and computer science student realized there are programming styles other than imperative and object-oriented programming. Namely functional programming paradigm. The lack of a study material dedicated to the functional programming withing the ITB Computer Science course curriculum and my curiosity toward the subject I decided to research and to learn the paradigm. I decided to use the course final project as the vehicle for this endeavour.

This paper is discussing functional programming paradigm with Scala programming language and web application development process using Play! Framework. The target audience is any computer science student with knowledge of a programming language such as Java and with an understanding of some of the object-oriented programming principles such as inheritance and polymorphisms.

List of Figures

Contents

Chapter 1

Introduction

1.1 Introduction

In the world of multi-core processors, distributed systems and big data, we are witnessing a paradigm shift in the computational model used within application development industry. It's a shift from imperative programming style toward declarative programming and the rise of the functional programming languages [**?**]. JVM ecosystem is no exception.

Even though Java being still most used JVM language and one of the most used programming languages overall, JVM languages such as Scala, Groovy and Clojure are getting more popular every day. Java itself with recent SE 8 update brought limited support for functional programming as well.

Scala programming language [**?**] is getting a big momentum in the last few years [**?**]. Companies such as Twitter or LinkedIn [**?**] had switched to Scala for their web applications development. Scala is being a big player in the realm of Reactive

programming, Distributed Systems and Big Data [**?**].

1.1.1 Some of the reasons for Scala success

- Scala is elegant, scalable, purely object-oriented and fully functional programming language with the performance comparable to Java.
- Scala compiles to JVM byte code .class files which mean that Scala is fully compatible with existing Java libraries or frameworks and vice versa. Mixed Java / Scala projects are quite common.
- Existence of powerful, open source and award-winning frameworks [**?**], toolkits, platforms and build tools written in Scala, such as Akka, Spark, Play, SBT [**?**].

1.2 Aims and Objectives

The idea is to develop a web application for a computer science student who would like to learn Scala programming language and functional programming paradigm. The application will be developed using MVC Play! Framework. I decided to work on this project for the following reasons:

- Strong personal interest in functional programming.
- Lack of modules dedicated to functional programming in ITB curriculum.
- Solid background in Java platform.
- Desire to pursue a Java / Scala development professional career.

1.2.1 Web Application Proposed functionality

- A student can register the account.
- The authenticated student will gain access to the functional programming with Scala lectures and exercises.
- The student will have access to Scala interpreter where he could carry out coding exercises.
- The student will be able to communicate with other users through a chat window or other messaging functionality.
- The student will be able to upload the files, for example an exercise solution.

1.3 Main Research Questions

- What exactly is functional programming and what benefits does it bring to application development?
- In what ways Scala supports the paradigm?
- What exactly is Play framework and what kind of applications we can develop with it?
- What are the advantages and disadvantages of Play framework in terms of productivity, performance and scalability and in comparison to other frameworks?

1.4 Justifications / Benefits

As mentioned earlier, there is not a module dedicated to functional programming in ITB Computer Science course curriculum. I can actually claim that there is no mention of this programming style in any module overall and the paradigm is totally ignored by the curriculum. In comparison, there are 8 networking modules delivered to students over the course duration. In my opinion, this makes the course unbalanced and creates an 'educational gap' in the curriculum. Personally, I feel that the topic belongs to the course curriculum and it is the natural progression from object-oriented programming and design patterns which had strong coverage in the third year of the course.

• **Filling the 'educational gap' in the Computer Science course curriculum by creating learning web resource.**

The preliminary research on the current state of the application development industry is suggesting that fluency in a functional programming language is a really valuable skill for any developer to possess right now. Developers able to code using languages such as Scala, Clojure, F# or Lisp dialects are in demand. Reasoning about the functional programs requires different 'mind-set' or 'thought processes' in comparison with object-oriented programming. Even if one would never use purely functional programming professionally, understanding of the concepts will make one a better programmer overall and give one deeper insight into other programming styles.

• **Acquiring valuable skills and deeper insides into different programming paradigms and design patterns.**

Also, Scala seems to be very elegant language and takes the good design ideas from many other programming languages. The language was designed with the scalability

of the syntax taken in the consideration, which allows the language to 'shape-shift' towards the needs of its users. In my opinion, the Scala programming language would be the right tool for me to learn functional programming paradigm.

• **Learning Scala programming language and Java Virtual Machine (JVM).**

Even though learning functional programming with Scala is the main aim of this project, the additional work will be carried out by exploring Play framework and developing the web application. The preliminary research suggests the framework is a powerful tool for rapid web development. Play seems to be heavily inspired by Ruby on Rails. It supports reactive web application development and seems to be a cleaner alternative to legacy Java Enterprise stack [**?**].

• **Building valuable skill set in contemporary web application development.**

1.5 Feasibility

To carry out this project work the following requirements must be met:

- Access to relevant study material and tutorials dedicated to Scala programming language and Play. There is no material available in ITB library, but there is a number of books published and available over the internet. *Estimated cost is 100 euros.*

- Access to personal computer. I'm the owner of the laptop computer, which should serve as the project workstation without too many constraints. Additional

research must be done to decide on the operating system and other tools. Using free software is most desired.*Estimated cost is 0 euros.*

- Access to the internet, deployment, DNS and versioning control services, printer and thesis binding. *Estimated cost is 300 euro.*
- At least 14 hours per week to carry out the project related work for the whole project duration. This estimation is based on personal experience I gained by working on two academic projects during my studies and on experience working on a research project as part of the summer internship.

In my opinion, all the requirements could be met to assure this project feasibility with the minimal cost associated. With proper planning, use of the personal assets and free software when possible, I should be able to meet the project goals within the given time scale.

1.6 Proposed Methodologies

Literature Review

The first step is to carry out the research on functional programming [**?**], Scala programming language [**?**], MVC design pattern and Play Framework [**?**]. I'm planning to use dedicated books I acquired recently, plus online tutorials and blogs. I will analyze, summarize, and carry out the coding tutorials. The thorough literature review at the beginning should lay a foundation for following web application development. I will continue to research relevant materials for the whole duration of the project.

The Web Application Development

The real challenge lies in the fact that I have a little experience developing web

applications using MVC framework and no skills in technologies I decided to use during this project. I have only general ideas how to design such an application. Only layers I can design reasonably well in advance are the database relations and user interface.

Therefore, I must approach this problem using some kind of adaptive methodology and avoid 'a big design in advance' approach. I would argue that Prototyping SDLC is the best methodology to carry out the development. In this model, the developer basically re-analyses, re-designs, re-implements and re-test application prototype until the product is accepted by the client.

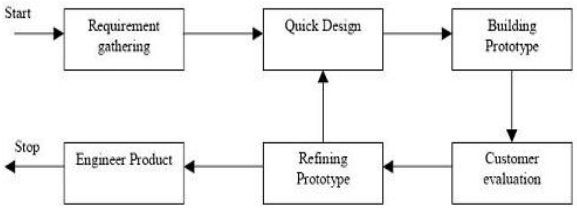

Figure 1.1: *Prototyping SDLC diagram*

Instead of creating 'a big design in advance', prototyping methodology is allowing the developer to change the design with each iteration of the design / development / evaluation / refining circle. I should expect major changes in design based on my lack of experience in web development and technologies I decided to use. The prototyping methodology will allow me to have a simple, but functioning prototype reasonably early. As more insights are acquired from the research, the prototype can be re-designed to implement additional functionality or re-evaluate the design.

1.7 Project Plan

1.7.1 Work Breakdown Structure (WBS)

1. **Research.**
   (a) Research on Scala programming language design ideas and syntax, MVC design pattern, Play framework, user interface and database design.
   (b) Literature Review of papers dedicated to the functional programming paradigm and test driven rapid web application development with Play.
   (c) Building development platform.
2. **Front-End design.**
   (a) Creating wire-frame design for new view.
   (b) Constructing the view.
3. **Database Design.**
   (a) Adding the database entity for new view.
   (b) Designing the relationships with existing entities.
4. **Prototype Development.**
   (a) Creating the controller object.
   (b) Creating unit tests.
   (c) Implementing method bodies using test driven development methodology.
   (d) Repeating the steps 1, 2, 3 with additional functionality until the final product is build.
5. **Quality Assurance.**

(a) Performing integration tests of the whole MVC pipeline.

(b) Performing user tests.

6. **Post Implementation Maintenance.**

(a) Making prototype production ready and deploying the prototype.

(b) Monitoring deployed application and adding more content.

(c) Adding content to the application.

(d) Working on project Report.

(e) Repeating steps 5.(a), 5.(b), 6.(a), 6.(b), 6.(c), 6.(d)

(f) Report Binding.

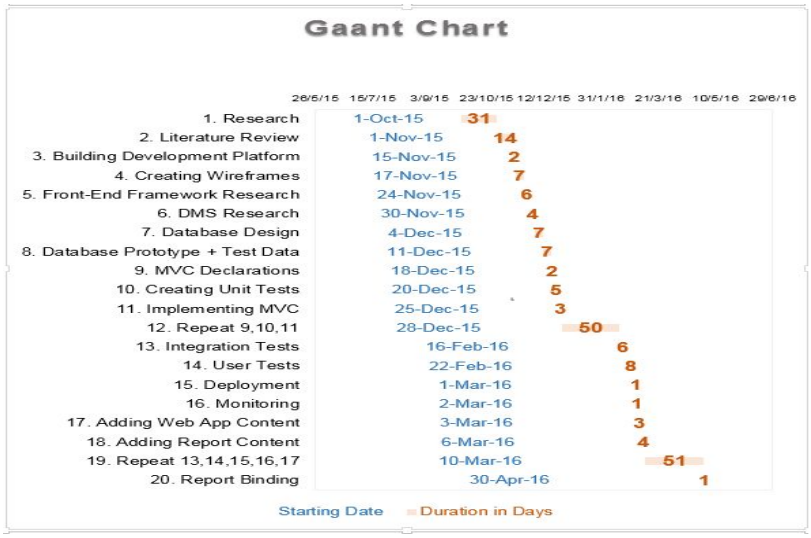

Figure 1.2: *Gaant Chart*

1.8 Expected Results

The project would be considered successful if it would meet at least two following criteria and goals. In the case of three or more goals accomplished I would consider project outcome to be very successful.

1. The gain in Scala coding skills and creation of enough learning material content for Scala course web application.

2. The comprehensive research on the functional programming paradigm and design patterns performed and gained the ability to code in the declarative style.

3. The delivery of fully functioning prototype of Scala course learning web application.

4. The acquisition of skill-set in rapid web development using Play framework.

1.9 Conclusion

At the end of this project, I will hopefully introduce some of Scala dedicated learning material in the form of an interactive web application. It can be used in a fictional course module for a fictional college. The application main purpose is to promote the interest in Scala programming language and functional programming paradigm. Especially on the personal level. After three years of my studies with ITB, I didn't learn anything about functional programming. I chose this project so I can gain a knowledge and skills which are really missing in my skill-set and which could prove very useful in the future. I hope as a fourth year student I am ready to face this challenge.

This research project will not contribute at all to the discipline area. Maybe only in a sense that it will hopefully bring one more student with the passion for programming languages to the functional programming paradigm. And maybe if other students or lecturers will see how elegant and declarative functional programming really is, the contribution could be a bit more significant.

Chapter 2

Literature Review

2.1 Introduction

Computer programming is a fascinating and vast subject to study. Programming languages began as an attempt to translate the human language and the way of human thinking into the language of a computer.

The idea was to make it easier, more efficient for programmers to write the programs. With the evolution of computers, as the problems to solve became more complex, the programs became larger and more sophisticated. The programming languages quickly evolved into the forms commonly referred to as high-level programming languages. In these forms, the programming languages are hiding the internal hardware details and offer a higher level of abstractions allowing programmers to write the programs using familiar terms with an ability to model real life objects.

The purpose of this paper is to review some of the studies dedicated to concepts and design ideas behind the high-level programming languages. The research done in this

field is immense. It is out of the scope of this paper to cover every concept,paradigm, or language and rather study some of the most important concepts in general. Then I will continue to study research papers dedicated to two most popular paradigms - object-oriented and functional programming. I will try to identify and discuss their key concepts, strength, and weaknesses and outline their history in short. I chose reading materials with a secondary intention to lay down a foundation for further studies toward a deeper understanding of the paradigms and transitions between them.

2.2 General Concepts

In [**?**] author defined a programming paradigm as an approach to a computer programming based on a coherent set of principles or a mathematical theory. The purpose for a paradigm existence is to solve a specific type of a problem. Each paradigm consists of a number of concepts. Any programming language can support one or more paradigms and the language which support more than one paradigm is called the multi-paradigm programming language. Different languages can interpret the concepts of a paradigm differently and often the implementations differ from language to language as well. The author listed around 30 useful programming paradigms implemented by modern programming languages.

Author then identified the two most important properties which differentiate the programming paradigms:

1. *Observable nondeterminism* is when a program is not completely determined by its specification. In other words with each execution the same program can produce different results. Observable nondeterminism is caused by the run-time scheduler and its usual effect is a race condition, which is often used as the

synonym for Observable nondeterminism.

2. *Named state* is an ability of a program to store values in time. This ability is highly influenced by the paradigm it contains it.

As mentioned before, the programming paradigms are based on a number of concepts. Author identified four most important programming concepts:

1. **Record** is a data-structure. Every programming language should be able to work with records. Arrays, lists, strings, trees and hash tables are derived from records.

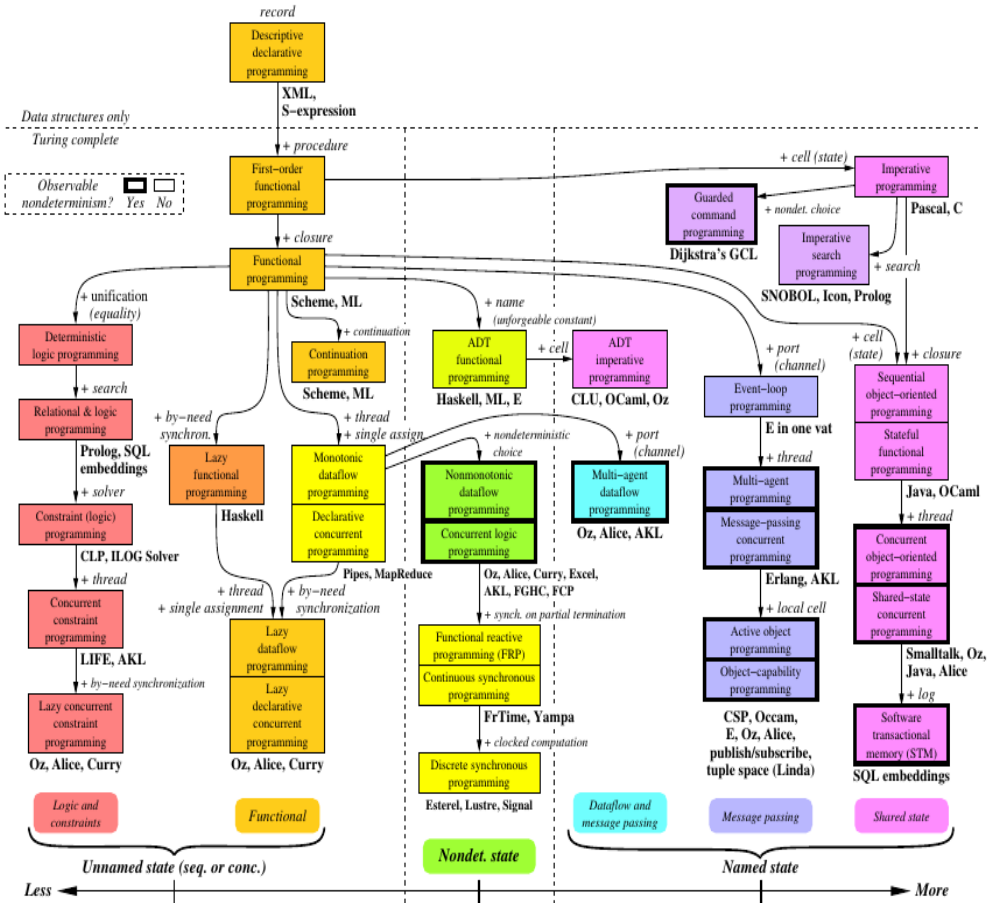

Figure 2.1: *Taxonomy of programming paradigms (Van Roy, page 15)*

2. **Lexically scoped closure** is a record storing a function together with an environment referenced at the time of that function definition. Closure is a very powerful concept and constructs such as *objects* or *control structures* have been implemented with closures in many programming languages.

3. **Independence** is when a program is constructed from independent parts. When parts don't interact with each other we call them *concurrent*. When an order of the execution is given, the parts are called sequential. The interaction between the parts is called communication. Author recognizes three levels of concurrency:

   (a) *Distributed system,* where concurrent activities (parts) are computers.

   (b) *Operating system,* where concurrent activities are processes. This level of concurrency is often called competitive concurrency because the processes are competing for access to system resources.

   (c) *Inter Process,* where concurrent activities are threads. This level of concurrency is called *cooperative concurrency* because the threads are cooperating to achieve the result of the process. There are two popular paradigms for inter process concurrency: *Shared state concurrency*, where shared data are accessed by threads using control structures called *monitors*. The second approach is *message-passing concurrency*, where threads are communicating by sending messages to each other.

4. **Named State** Author showed how named state could be implemented with a help of internal memory (variables).

2.3 Specific Characteristics

2.3.1 Object Oriented Programming

In [?] authors argued that consensus on what key concepts of Object Oriented Programming still don't exist. They argue that the most fundamental concepts behind the paradigm are as follows:

1. **Class** is a mechanism which group together attributes and methods with common properties. The class describes the run-time behavior of the objects instantiated from it. The well-designed class would have an immutable interface clients can use.

2. **Abstraction** is a simplified view of reality. Presented to clients by class methods and attributes.

3. **Inheritance** is a mechanism to create hierarchical class designs by creating a child class of the original class. A child class inherits the parent class behavior which can be then extended. Multiple inheritance allows for a class to inherit the behavior of multiple parent classes.

4. **Encapsulation** is hiding implementation details within the class. Only the interface is presented to the clients.

5. **Polymorphism** is allowing to similar looking structures to handle a variety of objects.

Authors then discussed inheritance and polymorphism in detail. I will outline the important key points.

Inheritance

As mentioned earlier the inheritance brings the hierarchical relationship into the class model. Authors pointed out that many languages define the most generic class that is an ancestor for all the classes in the language. This applies that any class can be downcast to the pointer of that ancestor. Authors pointed out that inheritance provides the ability to represent an *"is a"* relationship in software. This relationship can be violated if child class extends the inherited code the way which changes the code semantics. It is the reason for the inheritance to be used with caution.

Inheritance also allows to represent generalization /specialization relationships with method redefinition. This functionality is breaking the encapsulation because the child class has access to parent class hidden methods and attributes. This problem can be addressed by defining a well-defined interface for the descendants. Authors argued that literary every object-oriented language provides the ability for a child class to invoke parent class methods. Even those methods which were redefined in the child class.

Some of the languages support the feature which restrict method redefinition. A method is marked *'frozen'* or *'final'* and no child class can redefine this method. Authors warned that inheritance has also an adverse effect on synchronization requirements of a concurrent object. This problem is usually named *'inheritance anomaly'*. The problem arises when a class with concurrent code is derived. The careful redefinition of the inherited methods is necessitated to preserve the synchronization requirements. This necessity denies *'reuse'* benefits of inheritance. The multiple inheritance potential risks were then discussed in details. Authors explained various problems which could arise such as directed acyclic graph in a class hierarchy, method name collisions or repeated inheritance problem.

Authors outlined various solutions to deal with the problems and outlined the alternatives to multiple inheritance used by some of the object-oriented programming languages such as interfaces, mixins or delegation.

Polymorphism

As mentioned earlier, polymorphism allows programmers to write functions and classes which work uniformly with different types. Authors listed four distinct types of polymorphisms grouped into two categories. *Ad-hoc Polymorphism* and *Universal Polymorphism.* The difference between the categories is based on the fact that ad-hoc polymorphic functions execute code only for a small set of potentially unrelated types while universal polymorphic functions execute the same code for an infinite number of types.

Authors then defined the two types of ad-hoc polymorphism: The first type is *Overloading Polymorphism*, where the polymorphic function has the same name, but a different signature. Within this type authors distinguished between a *method* overloading and *operator* overloading and discussed each in detail.

The second type of ad-hoc polymorphism is *Coercion Polymorphism*. With coercion, the value of an argument can be converted to the value of another argument from the list of a method arguments. Authors pointed out that the difference between those two of ad-hoc polymorphism types is often blurry, especially in untyped and interpreted programming languages. The universal polymorphism is divided to two kinds as well. The first kind is *Parametric Polymorphism* aka generic programming. Generic programming uses type parameters to determine the type of a method argument. Authors pointed out that parametric polymorphism is only relevant for statically typed languages because dynamically typed languages infer types at run-time

and hence have generic programming built into them. The second kind is *Inclusion Polymorphism*, which gives different classes the ability to handle the same functionality.

Inclusion Polymorphism is what we call *Inheritance* in object oriented programming. Inclusion polymorphism is implemented through *dynamic binding* or sometimes called *late binding* because a method is bind to the message at the run-time. It is relevant for situations when a child class has an overridden method of the parent class and it is not obvious which method is being invoked. The search for the right method is then performed by the compiler (interpreter) at run-time. The dynamic binding uses the most specific version of a method.

Authors mentioned that some of the languages implement static binding where methods are bind to messages at compile time and it will always bind to a base class method .version.

2.3.2 Functional Programming

In [**?**] the author distinguished four key characteristics of modern functional programming languages.

Higher-Order Functions

In functional programming, the functions are treated as *'first class values'*, which means that they can be stored in data structures, passed as other function arguments and returned as results. The author pointed out that the function is the primary abstraction mechanism over values. He showed with examples how to compose higher-order functions.

Lazy Evaluation

Often called Non-Strict Semantics or call-by-need. Its primary feature is that arguments in a function call are evaluated at most once. In some cases, it does no evaluation at all. The author explained with the help of an example how lazy evaluation frees a programmer from concerns about evaluation order and pointed out the ability of the lazy evaluation to compute with infinite data-structures.

Data Abstraction Mechanisms

The author pointed out that a data abstraction improves modularity, security and clarity of programs. He explained that modularity is improved because one can abstract away from implementation. Data abstractions prohibit interface violations which improves security and that programs are clearer because of the self-documenting quality of the data abstractions. He argues that *strong static* typing eliminates type violations and run-time errors. He continued the discussion by describing *algebraic* (concrete) data types, *type synonyms*, and *abstract data types*.

Pattern Matching

The lack of side-effects in functional programs allows to apply pattern matching or sometimes called equational reasoning. The author explained the basics behind the feature with the help of code examples and outlined some of pattern matching shortcomings.

In (Hughes, 1984) author summarizes functional programming characteristics and advantages as they are usually used in the literature as follows:

- The functional program consists of functions.
- The functional program uses no mutable variables which, in general guarantee that program contains no side-effects. A program without side-effect is free of a major source of bugs.
- Since a function call produces no side-effects and for given arguments produces the same calculation result independently on when it is evaluated, the order of execution is irrelevant and functional programs are referentially transparent. This freedom makes functional programs easier to reason about.

The author argues that this often used list of strengths of functional programs is describing what functional programming is not (no assignments, no side-effects, no flow of control) and fails to emphasize what functional programming actually is. The list fails to emphasize the modularity as probably the most powerful characteristic and advantage of functional programs. He convincingly argues the ability of higher-order functions and lazy evaluation to increase the modularity by serving as a 'glue' for the program fragments. He provides a number of examples with code to support his claims.

2.4 Short History

First generally accepted object-oriented programming language is Simula (1967). With the introduction of Smalltalk (1962-1980) the paradigm gained some momentum. Most of the concepts of object-oriented programming were implemented in Smalltalk. In early 1980 the concepts were integrated into C programming language and resulting language was called C++. In the 1990s, the similar language was developed called Java by sun microsystems. Java became soon one of the most popular object-oriented

languages. Then in 2000, Microsoft announced .NET platform and C# programming language. C# is in many respects similar to Java. [**?**]

Functional Programming is heavily influenced by lambda calculus invented by Alonzo Church in 1936. First of the programming languages implementing lambda calculus was Lisp specified in 1958. Next significant language in terms of contributions to functional paradigm was Iswim introduced by Peter Landin in 1966. Probably the first functional language which received wide-spread attention was FP specified in 1978. In mid 70's several research projects related to functional programming emerged in the UK. Specifically the work of Gordon, Milner, and Wadsworth. They developed ML programming language which brought the invention of the type system (Hindley-Milner Type System). In early 80's David Turner at the University of Kent developed three languages which most faithfully characterize "modern school" of functional programming: SASL, KRC and Miranda. [**?**]

In later 1990s and after 2000 the functional programming has gained a great momentum and penetrated mainstream programming. Haskell, F#, Clojure, Scala are some of the examples of functional programming languages. Many object-oriented languages added functional features and become multi-paradigm languages. Java, Python, C# are examples of such.

2.5 Interesting Ideas for Further Study

2.5.1 A Definitive Programming Language

In [**?**] the author is presenting four research projects, each trying to solve a very different problem, but all four project considered language design as a key factor to

achieve success. Turned out that programming languages invented in each project have very similar structure supporting same paradigms: strict functional programming, declarative concurrency, asynchronous message passing and global named state. The invented languages are Erlang, E, Distributed Oz and Didactic Oz. One could infer ideas for a design of 'perfect language' and it could be quite interesting to study those four programming languages.

2.5.2 Artificial Intelligence

In [**?**] the author of the paper is using examples of composing programs from lazy evaluated higher-order functions. The final example is the alpha-beta heuristic algorithm. This algorithm is often used in computer games to estimate how good a player position is in the game. The author used Miranda programming language syntax, but it would be very interesting to implement the algorithm in some other functional programming language such as Scala or Haskell.

2.5.3 LambdaFicator

Java 8 update brought a few functional programming features in the language. Namely functions as first class values, lambda expressions and closures. Also fluent Stream API which uses monads, lazy evaluation, and higher-order functions. In [**?**] authors presented the analysis, design, implementation and evaluation of LAMBDAFICATOR, the automatic refactoring tool, which converts old-style code prior to Java 8 update into the functional style. Namely tool does two refactorings:

- Anonymous inner classes to lambda expressions

- External iterators to internal iterators (from *for loops* to Stream API *higher order function chains*)

In the paper, authors discussed their motivations, outlined the implementation algorithm in the detail and determined the usefulness with a thorough evaluation. They applied the tool to four open source projects (ANTLRWorks, Apache Ivy, Junit, Hadoop) with very successful results. For example, the first type of refactoring reduced the number of source code lines by 2213 with 100 percent accuracy.

The tool is open source and available for download and would be very interesting to study it deeper and implement it using some other programming language such as Scala or Haskell.

2.6 Conclusion

The main purpose of this research was to study two most popular programming paradigms: object-oriented and functional programming. To identify and understand the key concepts and to find the similarities and differences. The first step was to understand general programming concepts such as record, closure, state and concurrency. Then I focused on studying characteristics of object-oriented programming languages. I learned that main building blocks of object-oriented programs are classes organized into hierarchies based on inheritance. I learned about polymorphism, the powerful feature of programming languages in general. Then I studied papers dedicated to functional programming and I found out that functional programs are built from functions, which are then composed together to modules and programs. I was shown the elegance of lazy evaluation and pattern matching. I learned that functional programs are trying to eliminate state and to stay referentially transparent.

Chapter 3

Analysis and Design

3.1 Introduction

In this section, I will outline the content of the current chapter in a single paragraph. I will include short overview of a web application design with a diagram

3.2 Proposed Methodology

I decided to use Prototyping methodology developing this web app, so I will discuss the reasons for the decision and will outline the development process.

3.3 Proposed Tools

In this section, I will list all the development tools and technologies I will use within this project.

3.4 Use Cases

This section will contain use case scenarios and UML use case diagrams.

3.5 Sequence Diagrams

This section will contain sequence diagrams for particular use cases.

3.6 User Interface Design

This section will contain proposed wire-frames for user interface views of the web app.

3.7 Database Schema Design

In this section I will discuss the proposed database schema design with help of ERD diagram.

3.8 Back End Design

This section will discuss the decision of using Scala and Play to develop this application, while identifying an advantages and disadvantages.

3.9 Conclusion

In this section I will conclude Analysis and Design chapter.

Chapter 4

Implementation

4.1 Introduction

In this section, I will outline the content of the current chapter.

4.2 Methodology

In this section I will document the development process in form of prototype snapshots.

4.2.1 Prototype v1.0

In this section, I will list and discuss all the functionality improvements and changes implemented in this prototype. With help of code snippets, tables and figures.

4.2.2 Prototype v2.0

In this section, I will list and discuss all the functionality improvements and changes implemented in this prototype. With help of code snippets, tables and figures.

4.2.3 Prototype ...and so on

In this section, I will list and discuss all the functionality improvements and changes implemented in this prototype. With help of code snippets, tables and figures.

4.3 Conclusion

In this section, I will conclude the Implementation Chapter.

Chapter 5

Testing and Evaluation

5.1 Introduction

In this section, I will outline the content of this chapter. I will also discuss the types of tests, test designs and testing frameworks and tools.

5.2 Back End Unit Tests

In this section, I will discuss unit test performed on server side code base with help of code snippets and figures.

5.3 User Interface Tests

In this section, I will discuss the automation testing of user interface.

5.4 Conclusion

In this section, I will conclude the Testing and Evaluation Chapter.

Chapter 6

Conclusion and Further Work

6.1 Introduction

In this section, I will outline the content of this chapter.

6.2 Achievements

Here I would summarize the project achievements.

6.3 Personal Gain

In this section, I will list all the experience and knowledge I have acquired by working on this project.

6.4 Further Work

This section content will be listing of all possible improvements and further work in terms of security, optimization and user interface design.

6.5 Conclusion

In this section, I will conclude current chapter.

Appendix A

The Project Diary

Here will go the project diary in a form of week by week work listings organized in table formats.

Appendix A

Code Listings

A.1 Back-End

This section content will be all the server side Scala/Play code.

A.2 Front-End

This section content will be all the user views (web-pages) code.

Bibliography

[1] Neil Ford, *Functional thinking: Why functional programming is on the rise*, http://www.ibm.com/developerworks/library/j-ft20/ (2013-01-25)

[2] Typesafe, *Scala, Object-Oriented Meets Functional*, http://www.scala-lang.org/, (2002-2015)

[3] Typesafe, *Scala in the Enterprise*, http://www.scala-lang.org/old/node/1658, (2012-01-19)

[4] Brikman, Yevgeniy, *The Play Framework at LinkedIn: Productivity and Performance at Scale*, https://www.youtube.com/watch?v=8z3h4Uv9YbE&ab_channel=NewCircleTraining, (2013-06-26)

[5] Wampler, Dean, *We Won! How Scala Conquered the Big Data World.*, https://www.youtube.com/watch?v=AHB6aJyhDSQ&ab_channel=NewCircleTraining, (2015-03-01)

[6] theotown, *Akka wins 2015 JAX Award for Most Innovative Open Source Technology*, https://www.typesafe.com/blog/akka-wins-2015-jax-award-for-most-innovative-open-technology, (2015-03-23)

[7] Wikipedia, *Free software programmed in Scala*,
https://en.wikipedia.org/wiki/Category:Free_software_programmed_in_Scala, (2015-01-05)

[8] Wikipedia, *Play Framework*,
https://en.wikipedia.org/wiki/Play_framework, (2015-10-04)

[9] Paul Chiusano, Runar Bjarnason, *Functional Programming in Scala.*,
Shelter Island, NY 11964, Manning Publications Co., 2015. ISBN 9781617290657.

[10] Dean Wampler, Alex Payne, *Programming Scala, Second Edition.*,
1005 Gravenstein Highway North, Sebastopol, CA 95472, O'Reilly Media, Inc., 2015. ISBN: 978-1-491-94985-6.

[11] Peter Hilton, Erik Bakker, Francisco Canedo, *Play For Scala.*,
Shelter Island, NY 11964, Manning Publications Co., 2014. ISBN 9781617290794.

[12] Hudak, P., *The Conception, Evolution, and Application of Functional Programming Languages.*,
s.l.:Yale University Department of Computer Science, 1989

[13] Hristakeva, M., Vuppala, R., *A Survey of Object Oriented Programming Languages.*,
Santa Cruz: Univ. of California. 2009

[14] Hughes, J., *Why Functional Programming Matters.*,
Goteborg Institutionen for Datavetenskap, Chalmers Tekniska Hogskola, 1984

[15] Goyri, A., Franklin, L., Dig, D., Lahoda, J., *Crossing the Gap from Imperative to Functional, Programming through Refactoring.*,
Saint Petersburg: s.n., 2013

[16] Roy, P. V., *Programming Paradigms for Dummies: What Every Programmer Should Know.*,
s.l.:IRCAM/Delatour, 2009