Exploring Rapid Application Development for Android with Scala and SBT

Luís Miguel Guimarães Pimentel Fonseca

Master in Informatics and Computing Engineering

Supervisor: Hugo Sereno Ferreira, PhD
Co-supervisor: Tiago Boldt Sousa

June 2014
Exploring Rapid Application Development for Android with Scala and SBT

Luís Miguel Guimarães Pimentel Fonseca

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Doctor João Manuel Paiva Cardoso
External Examiner: Doctor Ricardo Jorge Silvério Magalhães Machado
Supervisor: Doctor Hugo José Sereno Lopes Ferreira
June 2014
Abstract

Android development, despite currently widely adopted, is still troublesome as developers need to learn very complex Application Programming Interfaces to start implementing their projects. Furthermore, the Java programming language presents design issues that have been addressed in modern languages, which also does not ease the complexity in Android development.

Following a paradigm widely used in web development, Model-View-Controller, frameworks can simplify development, providing an abstraction layer in order to create complex applications.

In this dissertation, we present an approach to simplify the development process of an Android application through the adoption of the patterns commonly found on web development, along with the introduction of the Scala programming language instead of Java. Scala Build Tool was put to work as a build manager. Our approach consisted on creating an umbrella project for other already existing tools alongside new ones, motivating the best practices while using them together.

We start by gathering background on web patterns that could help rapid prototyping and that could be employed in Android. With these notions, a broad analysis of the current state of the art was done, with focus on Web frameworks. Furthermore, we also took a look at the existing Android projects that are relevant to our work. Then, we searched for existing tools that could become dependencies of the solution.

These ideas, patterns and plugins come together in a complete and usable framework to support Android application rapid development and prototyping with Scala. The features implemented are discussed in detail and their impact in the workflow is measured.

Finally, the framework is validated through an empirical controlled experiment performed with students. This allowed us to assess the positive impact the framework had and to prioritize possible future work.
Resumo

Apesar de o desenvolvimento de aplicações Android ser bastante comum, este é de difícil aprendizagem, visto que, para implementarem os seus projectos, os programadores se deparam com uma Application Programming Interface complexa e extensa. Acresce ainda o facto de, a linguagem de programação Java, apresentar problemas estruturais que já foram colmatados em linguagens modernas, o que também não contribui positivamente para a complexidade no desenvolvimento em Android.

Nesta dissertação é apresentada uma abordagem que procura simplificar o desenvolvimento de aplicações Android, através do uso de padrões tipicamente encontrados no desenvolvimento de aplicações Web, recorrendo do uso de uma linguagem moderna — Scala. Foi utilizada a ferramenta Scala Build Tool como build manager, apresentando-se uma solução final na qual foram integrados projectos relacionados que facilitam o desenvolvimento em geral. Estamos convictos de que esta abordagem proporciona um ambiente de desenvolvimento de aplicações Android mais ágil.

Começámos por analisar quais o padrões da Web que poderiam ser aplicados no desenvolvimento de aplicações Android capazes de potenciar prototipagem rápida. Posteriormente, avançámos para uma pesquisa bibliográfica extensa e alargada, com especial atenção nas Web Application Frameworks. Não obstante, analisámos também projectos Android relevantes para o nosso trabalho. Por último, procurámos encontrar ferramentas previamente existentes passíveis de fazer parte da nossa solução.

Estas ideias, padrões e ferramentas, foram reunidos numa plataforma completa e estável, por forma a promover o desenvolvimento e prototipagem rápida de aplicações Android com Scala. As funcionalidades adicionais implementadas foram discutidas em detalhe e avaliado o seu impacto no workflow do desenvolvimento de aplicações.

Por fim, procedeu-se à validação da ferramenta através de um estudo empírico, que assume a forma de uma experiência sintética com estudantes, num ambiente controlado. Isto permitiu-nos compreender, não só o impacto positivo que a ferramenta tem, mas também como priorizar o trabalho futuro.
Acknowledgements

I would like to express my gratitude to Professor Hugo Sereno Ferreira, my supervisor, for sharing his constructive views and insights during the planning and development of this dissertation. His dissertation also greatly inspired my report in the experiment data analysis. Likewise, I would like to acknowledge Tiago Boldt Sousa, my co-supervisor, for his constant support and guidance when I most needed. Both exchanged valuable ideas with me and still gave enough room to freely experiment my own.

Special thanks should be given to my fellow friends, who are also currently finishing their masters degree, for their support and friendship: Diogo Castro, Omar Castro, Ricardo Amorim and Sara Morais. Additionally, I would like to express my deep appreciation to Gaspar Furtado who took countless hours in reviewing my English.

Finally, I would also like to extend my gratitude to my family, my girlfriend and closest friends for their companionship and emotional support: Ana Pestana, Joaquim Silva, João Rui, Raquel Segadães, Tiago Rodrigues and Vanessa Pinto.

Luís Miguel Fonseca

---

1Perhaps my gratitude would not be complete if I did not acknowledge Bach, Beethoven and Mozart for the meticulously well-crafted musical compositions which assisted me throughout the course of the dissertation.
“The best way to predict the future is to invent it.”

Alan Curtis Kay
Contents

Abstract  i

Resumo  ii

1 Introduction  1
1.1 Context  1
1.2 Motivation  3
1.3 Main Goals  3
1.4 Outline  4

2 Background  5
2.1 Functional Programming and Scala  5
2.2 Patterns of Web Development  6
  2.2.1 Model-View-Controller pattern  6
  2.2.2 Data Access Object pattern  8
  2.2.3 Active Record pattern  8
2.3 Development Processes Patterns in Web Application Frameworks  9
  2.3.1 Database Management  10
  2.3.2 Model Generation  11
  2.3.3 Scaffolding  11
  2.3.4 Continuous building  12
2.4 Android  13

3 State of the Art  15
3.1 Android  15
  3.1.1 Android Studio IDE  16
  3.1.2 Android Developer Tools  17
  3.1.3 Scaloid  17
  3.1.4 AppDF  19
3.2 Web Application Frameworks  19
  3.2.1 Play!  19
  3.2.2 Ruby on Rails  20
  3.2.3 Django  21
3.3 Build tools  22
  3.3.1 Maven  22
  3.3.2 Simple Build Tool  23
3.4 Existing SBT Plugins  23
  3.4.1 jberkel/android-plugin  23
4 Rapid Prototyping and Development Framework
   4.1 Overview ............................................................... 27
   4.2 Framework Functionalities ........................................ 28
      4.2.1 New Project .................................................. 28
      4.2.2 Model generation ............................................. 30
      4.2.3 Scaffold ...................................................... 32
      4.2.4 Database support ............................................ 34
      4.2.5 Automatic permission resolution .......................... 36
   4.3 A tutorial approach ............................................... 37
   4.4 Development Effort ................................................ 37
   4.5 Conclusions ......................................................... 38
5 An Academic Quasi-experiment ........................................ 39
   5.1 Research Design .................................................... 39
      5.1.1 Methodology ................................................ 40
      5.1.2 Treatments ................................................... 40
      5.1.3 Pre-test Evaluation ......................................... 41
      5.1.4 Process ....................................................... 41
      5.1.5 Post-test Evaluation ....................................... 42
      5.1.6 Questionnaires .............................................. 42
   5.2 Experiment Description ............................................ 42
      5.2.1 Preamble ...................................................... 43
      5.2.2 Task 1 — Generate a new project ......................... 43
      5.2.3 Task 2 — Implement Models ................................. 43
      5.2.4 Task 3 — Implement Views and Controllers ............. 43
      5.2.5 Task 4 — Use a database to store Products .............. 44
      5.2.6 Task 5 — Extra functionality ............................. 44
   5.3 Data Analysis ........................................................ 44
      5.3.1 Background .................................................. 45
      5.3.2 Tutorial ....................................................... 46
      5.3.3 External factors ............................................. 47
      5.3.4 Overall satisfaction ........................................ 47
      5.3.5 Development process ....................................... 49
      5.3.6 Usability Satisfaction ..................................... 50
   5.4 Objective Measurement ............................................ 52
      5.4.1 Duration of tasks ........................................... 53
      5.4.2 Graphical User Interface completeness .................. 54
      5.4.3 Application startup time .................................. 55
      5.4.4 Runtime crashes due to lack of permissions during development .............................. 55
   5.5 Validation Threats .................................................. 56
   5.6 Conclusion ............................................................ 56
CONTENTS

6 Conclusions and future work 59
   6.1 Overview ......................................................... 59
   6.2 Contributions .................................................... 59
   6.3 Future work ....................................................... 60
   6.4 Final Remarks .................................................... 60

A Typesafe’s Activator Tutorial 67
# List of Figures

2.1 Model-View-Controller State and Message Sending ........................................ 7  
2.2 DAO pattern in UML .................................................................................. 9  
2.3 Active Record pattern in UML .................................................................... 10  
4.1 Architecture overview with external plugin dependencies. ....................... 28  
4.2 New Project Android sequence diagram. ..................................................... 29  
4.3 Generate a new Model sequence diagram. ................................................. 31  
4.4 UML diagram representing the model schema. .......................................... 32  
4.5 Generate a new Model sequence diagram. ................................................. 33  
4.6 Screenshots of the resulting application. .................................................... 34  
4.7 A diagram exemplifying the process of generating the database conversion. ... 35  
4.8 Evolution of the `agile-scala-android` plugin development ....................... 38  
5.1 Experiment design activity diagram ........................................................... 40  
A.1 Screenshot of the *Getting Started* tutorial section. ............................... 68  
A.2 Screenshot of the *New (Scala) Project Android* tutorial section. ............ 69  
A.3 Screenshot of the *Creating a new Model* tutorial section. ...................... 70  
A.4 Screenshot of the *Database* tutorial section. ......................................... 71  
A.5 Screenshot of the *Automatic Android Permission resolution* tutorial section. 72  
A.6 Screenshot of the *Scaffolding* tutorial section. ....................................... 73  
A.7 Screenshot of the *Wrap up* tutorial section. .......................................... 74
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Feature analysis comparison between the several technologies</td>
<td>26</td>
</tr>
<tr>
<td>5.1</td>
<td>Grouped student current grades statistics</td>
<td>41</td>
</tr>
<tr>
<td>5.2</td>
<td>Succinct overview of the Background results, containing the non-parametric Mann-Whitney-Wilcoxon test.</td>
<td>45</td>
</tr>
<tr>
<td>5.3</td>
<td>Succinct overview of the Tutorial results, containing the averages and standard deviations.</td>
<td>46</td>
</tr>
<tr>
<td>5.4</td>
<td>Succinct overview of the External Factors results, containing the non-parametric Mann-Whitney-Wilcoxon test.</td>
<td>47</td>
</tr>
<tr>
<td>5.5</td>
<td>Succinct overview of the Overall Satisfaction results, containing the non-parametric Mann-Whitney-Wilcoxon test.</td>
<td>48</td>
</tr>
<tr>
<td>5.6</td>
<td>Succinct overview of the Development Process results, containing the non-parametric Mann-Whitney-Wilcoxon test.</td>
<td>49</td>
</tr>
<tr>
<td>5.7</td>
<td>Succinct overview of the Usability Satisfaction results, containing the averages and standard deviations.</td>
<td>51</td>
</tr>
<tr>
<td>5.8</td>
<td>Succinct overview of the duration of tasks in minutes.</td>
<td>53</td>
</tr>
<tr>
<td>5.9</td>
<td>Overview of the Graphical User Interface completeness.</td>
<td>55</td>
</tr>
<tr>
<td>5.10</td>
<td>Overview of the runtime crashes experienced by the participants during development.</td>
<td>55</td>
</tr>
</tbody>
</table>
LIST OF TABLES
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AOT</td>
<td>Ahead-of-time</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update and Delete</td>
</tr>
<tr>
<td>DAO</td>
<td>Data Access Object</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>OMR</td>
<td>Object-Relational Mapping</td>
</tr>
<tr>
<td>SBT</td>
<td>Simple Build Tool</td>
</tr>
<tr>
<td>TDD</td>
<td>Test Driven Development</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Context

Android is currently the most popular multi-purpose operating system with the highest market-share in smartphones worldwide, reaching almost 80% [Gar] and it is not showing any signs of losing ground to any of its competitors either [Str]. This large user base translates into a high demand for companies and freelancers to build quality applications, having over one million been published on Google Play Store [Gsm]. The applications are diverse in design and functionality, the competition is fierce and the speed of deployment is increasingly important; however, the process to create an application from start to finish, is never a simple one and requires deep understanding of the Android platform beforehand. In the end, this constrains the time to market factor for those not experienced with the Application Programming Interface (API).

Android was initially designed to run on smartphones, but later evolved to tablets and to other device formats. As this evolution took place, its API became more and more extensive\(^1\) and the learning curve to use it has become increasingly steep.

A good Integrated Development Environment (IDE) can and should tackle these shortcomings mostly by providing documentation, templates and suggestions on how to solve common problems. However, the shortcomings of the Java programming language are not addressed directly. These shortcomings are mostly due to the current lack of progress over the years. For example, Java does not support functions as first-class citizens, which led to many interfaces that only require one method to be implemented — in all of these situations, passing a function would suffice.

Scala is a multi-paradigm object-oriented programming language that also compiles to the same Java byte-code, and thus runs on any Java Virtual Machine (JVM)\(^2\) [OAC+04]. Since Android systems also have their own implementation of a JVM, Scala is just as capable as Java on

---

\(^1\)In 2011 Android reportedly had over 1 million lines of code [All11].

\(^2\)There was also a Scala variant that run on the .NET platform, but official support has been dropped.
running in this environment. Moreover, Scala is fully interoperable with Java — this ultimately is one of the very reasons that make it increasingly popular in the industry [O’G14].

We can argue that this fundamental change in the language choice brings to the developer considerable benefits:

- **Simpler Syntax and Abstractions** — Scala’s syntax expressiveness can be leveraged to abstract Android’s API and write more concise code. Scaloid, a Scala library for Android projects, makes use of these powerful abstractions on top of the Android API. Using Type Inference, Mixins and Implicits\(^3\), one can shorten the necessary code and possibly lower the needed development time whilst increasing the maintainability.

- **Functional programming paradigm** — Code in functional programming languages tend to be less complex than the imperatives counterparts [FASS96]. A major benefit from this paradigm is writing concurrent code. Not having global state is a much more direct way to handle concurrency since the programmer is not required to manage the access of mutable variables to prevent dead-locks or race conditions. The philosophy is to program in terms of space and object construction — which data needs to be computed to perform the next computation — as opposed to program in terms of time — when is the data ready to perform the next computation. With the number of devices with multiple cores growing, this benefit is harder to overlook. To this matter, Scala language designer, Martin Odersky, advocated the following:

  “Being functional, Scala sheds a lot of the liabilities of imperative code, which becomes very, very serious as you go parallel.” [Dob, Martin Odersky]

- **No runtime performance penalty** — Although not necessarily a benefit, we can safely switch from Java to Scala without compromising the runtime performance. Intuitively, we are inclined to think functional programming as inefficient. However, one empirical study was conducted to verify this hypothesis and concluded that the performance of Scala versus Java was very similar [PSG12]. Following the same premises, another study arrived to the same conclusions, with the testing being performed on an Android environment [DN13].

When analysing the development process of an Android application, we can argue that it is quite similar to the development process of Web Application Frameworks. This happens because both share the same core pattern Model-View-Controller (MVC) [KP88b]. This is a well known and proven pattern since it decouples the Model and the View from the Controller, respecting the single responsibility principle [Mar03].

In addition to sharing the core pattern, Web Applications Frameworks are more mature in the sense they almost always provide some form of abstraction to handle databases, contain a template engine, while also promoting code reuse when possible. Each programming language usually has

\(^3\)These language concepts are presented in detail in section 2.1.
Introduction

at the very least one framework that is well known and established: PHP has Lavarel\(^4\), Ruby has Ruby on Rails\(^5\), Python has Django\(^6\), and Java and Scala have Play\(!\)\(^7\).

These features appear in Android, in a somewhat more limited form. For example, Android contains a SQLite database backend, but implementation details is the full responsibility of the developers. The developed solution here presented shares the same approach of these frameworks, easing implementation while increasing quality and speeding up application development.

Is it possible that there are more patterns and development processes present in Web Frameworks that can be brought to Android development? This thesis aimed at providing a solid answer to this question.

1.2 Motivation

Since Scala is not a first-class citizen when it comes to Android applications, current support is somewhat limited, when compared to Java. There are several projects that ease the task too, but they lack functionalities and integration, they do not provide continuous development assistance or do not explore the other advantages a web framework provides through their console based management.

Another aspect to consider is Android’s extensive API that results in a complex learning process. This could even translate to a bigger problem when not using the Java language — at the very least, naming conventions will be broken. In this case, it is possible to leverage Scala’s expressiveness to create simpler abstractions while using the Android API.

The prospect of building such tool and have a real impact in the community was just too much to pass on.

1.3 Main Goals

The quintessential contributions of this thesis are the listed as follows:

1. **Research Web Application Frameworks processes and patterns they promote** — Development and design patterns from Web Application Frameworks were researched. The web development is overall more mature than mobile programming, mostly due to the fact that mobile programming is still a novelty.

2. **Android Rapid Prototyping and Development Framework with Scala** — The goal of building a framework that actually improves the development process and successfully creates a valid rapid prototyping environment allowed to further improve the research of patterns and development methodologies. This goal also allowed the possibility of performing an experiment to assess the validity of these concepts.

\(^4\)For more information consult [http://laravel.com](http://laravel.com).
\(^5\)For more information consult [http://rubyonrails.org](http://rubyonrails.org).
\(^6\)For more information consult [https://www.djangoproject.com](https://www.djangoproject.com).
\(^7\)For more information consult [http://www.playframework.com](http://www.playframework.com).
3. **Validation through experimentation** — An academic quasi-experiment was performed to assess and evaluate the usefulness of the developed framework.

4. **Result dissemination** — The research on development processes and patterns along with the creation of a real framework led to the writing of two paper submissions for Pattern Languages of Programs (PLoP)\(^8\) and INForum\(^9\).

In summary, can Scala’s expressiveness be leveraged to abstract Android API? Also, is it possible to create a framework to support rapid prototyping in this environment?

### 1.4 Outline

In this chapter, the context and motivation for this thesis was introduced. Asides from this introduction, this document contains five more chapters.

In the chapter *Background* (2) a literature review over functional programming, Scala programming language and web patterns are discussed. This is followed by the *State of the Art* (3) where an analysis of existing build tools, web frameworks and Android is made.

Next, *Rapid Prototyping and Development Framework* (4) describes the materialization of the concepts previously discussed in an actual usable framework. This framework was validated through *An Academic Quasi-experiment* (5) which was carefully analysed and documented.

Lastly, this document ends with *Conclusions* (6) presenting a succinct and broad overview of the contributions of this work, as well as discuss possible future work and draw some final considerations.

---

\(^8\)For more information on of this conference, consult [http://www.hillside.net/plop/2014/](http://www.hillside.net/plop/2014/).

Chapter 2

Background

2.1 Functional Programming and Scala

Functional programming exists much longer than computers themselves. Lambda calculus was first formulated by Alonzo Church to formalize the concept of effective computability in the late 1930s [OSV08]. It was only much later that these concepts appear as an actual language with Lisp, the first functional programming language, introduced to the world back in 1958. As time passed, the language never had much adoption in the industry and was mostly an academic construct. However, recent years have seen an increased interest in functional programming languages and techniques [O’G14].

There are two main building blocks to any functional programming language [OSV08]. Firstly, functions are first-class values. The idea behind this concept is that functions can be passed through arguments, a function can return another function and they can be stored in variables, much like storing a String or an integer. Secondly, the functions should map an input to an output, rather than change the state of the system.

The benefits of this paradigm are paramount when considering concurrent programming. As mentioned in the very first section of this document (section 1.1), the lack of a state is a much more direct way to handle concurrency. Dead-locks and race conditions are two typical problems of having to deal with state in a concurrent environment. Considering today’s hardware, with special attention to smartphones, a comparison between the high-end and low-end devices can be made to see the current trend in evolution; by doing so, this allows us to understand the number of cores is increasing and the programmers need to cope with this aspect.

The world as we know it, might not be quite as functional. Imperative programming still has its merits and usefulness and there are operations that are inherently mutable, such as Input/Output (I/O) or networking. Scala fuses the functional and the imperative paradigms to serve a wider spectrum of applications.

Some of the Scala’s characteristics and relevant features are discussed as follows:
Background

- **Traits** — Scala allows Classes to be extended from several Traits which can also have a default implementation. This promotes code reuse as several traits can be mixed in one Class.

- **Type Inference** — When omitted, the type of the variable will be inferred automatically when compiling. This means the programmer can assign, for example, integer 42 to a variable without specifying that the variable type is an integer.

- **Case Classes** — This is a Scala’s special kind of Classes. These build on top of regular Classes and contain a meaningful implementation of helper methods (toString(), hashCode() and equals() along with the fields getters and setters.) while also exporting their constructor arguments to support pattern matching.

- **Pattern matching** — Pattern matching like a Java switch, only much more powerful and capable of matching class hierarchies, sequences and more.

- **Implicit** — When there’s a mismatch between expected types and actual types, the implicit functions within the scope will be executed implicitly, converting the data to the required type.

- **Lazy evaluation** — Variables or expressions that only get evaluated when they are first accessed are being loaded lazily. In functional programming, having variables or functions without side effect can be executed in a lazy fashion as the order of execution is irrelevant.

It is relevant to note that none of these features are currently present in Java, which makes Scala a convincing argument as a valid alternative for Android programming.

### 2.2 Patterns of Web Development

In a broad sense, a pattern describes a recurrent problem which occurs within a given context and can be resolved with a known solution [AIS77]. This definition was formulated by Alexander in his book A Pattern Language and has since been used in computer science. Patterns can be simply considered as the triad of \(< problem, context, solution >\).

The next sections will discuss three important software design patterns typical found in web development\(^1\): Model-View-Controller [KP88a], Data Access Object [Ora] and Active Record [MVM10].

#### 2.2.1 Model-View-Controller pattern

**Problem**

If an end-user requests a webpage, what really happens from the beginning of the request to provide the final page? How can the functionality be separated from the structure of an application and the way it is represented?

\(^1\)These will be discussed in detail in the next chapter, section 3.2.
Background

Context

When developing web applications with a particular framework, developers will find themselves employing the Model-View-Controller pattern, as it is mostly enforced as part of the frameworks’ API.

Perhaps the context is not even web development, rather an end-user interactive application.

Solution

When building interactive applications, be it web applications or not, the modularity of components have enormous known benefits. By isolating dependencies as much as possible, and honouring the single responsibility principle, it makes it easier for the application designer to grasp the purpose of a particular unit, without having to worry about the rest. Code refactoring and testing is also easier as less-side effects can occur [KP88b]. The pattern MVC is applied at the core of a framework and it forces the programmers to follow it, even if unknowingly.

The figure 2.1 illustrates the information flow and components that form the pattern.

Figure 2.1: Model-View-Controller State and Message Sending [KP88a, slightly adapted]

Models

The Model component is the application’s domain-specific representation of the data.

Views

The View component is responsible for the visual and graphic aspects of an application; they display the data from the Model and are reactive to its changes.

Controllers

The Controller component acts as an interface between their associated models and views and the user input.
Background

When it comes to an Android application, this pattern also occurs. The Views are stored in independent XML files; the Controllers can be considered the activities; the Models can be simple classes that hold information and notify changes to observers. However, the MVC pattern advocated in Android can be considered strongly coupled since it is not possible to instantiate an Activity (the controller) without a View.

2.2.2 Data Access Object pattern

Problem

How can we change the database back-end without breaking the entire code base? Moreover, how can we dissipate the database intricacies and idiosyncrasies from the rest of the code? Let's take these questions into consideration for a moment. Now imagine an application where SQL code is scattered everywhere in the code logic. The slightest change in the database will require at least refactor *somewhere* in the application. Finally, besides the problem of having SQL scattered everywhere, let's subjectively consider how the average developer is at ease with SQL.

Context

The questions raised are valuable in highly dynamic projects where the database might be subject to change due to legal aspects, licensing costs or other reasons. Possibly changing the database is not expected to break the rest of the application as different people or teams work in parallel.

Solution

Data Access Object (DAO) is a design pattern that provides an abstract layer that exposes the database (or other persistence mechanism) and establishes a contract for the required operations. By mapping the database functions calls, a DAO can provide specific data operations without exposing the implementation details of the database, effectively decoupling database access code from other application logic [Ora].

Figure 2.2 illustrates how the pattern can be applied.

When applied correctly, this pattern nicely supports the Single Responsibility principle [Mar03]. If a change in the database is needed, the DAO only needs to guarantee that it still respects the previous established contract. Additionally, the details of the database are also (possibly) hidden.

2.2.3 Active Record pattern

Problem

Is it possible to avoid writing SQL while still having a complete database backend? Even more importantly, can we have a system implemented in such a way that it is seamlessly integrated with the programming language?
Background

Figure 2.2: DAO pattern in UML [vD13, slightly adapted].

Context

The database efficiency is not a top-priority concern. Development speed however is crucial, and the project needs to reach a usable state as soon as possible. Furthermore, the domain logic is not too complex.

Solution

The Active Record is accountable for unifying both the domain logic and how the data should be stored logic. It is a coupled solution where the data structure is also expected to match exactly that of the database (a class field will be a column in the table). Each Active Record has the responsibility to have all the needed functionality for saving and loading to the database. This however, could be generated automatically by some frameworks [MVM10].

With this pattern, the classes are convenient, but they don’t hide the fact that a relational database is present [Fow02].

Figure 2.3 illustrates how the pattern can be applied.

On argument against Active Record is the fact that it couples the object design to the database design. This in turn means that it is more difficult to refactor either design as a project goes forward.

2.3 Development Processes Patterns in Web Application Frameworks

A Web application framework is a framework designed specifically to help software developers to build better web applications or web services. They are usually dependent on (and built for) one specific language. By using a framework, a developer can avoid unnecessary re-implementations of the same features that might have been done by others in the past.

It is the author’s belief that web frameworks share a lot in common with mobile computing since they already share the same core pattern — MVC.
The features that each web framework has varies greatly. Nonetheless, they share the most essential ones. The following sections describe compelling patterns currently employed for developing web applications.

2.3.1 Database Management

Problem

Setting up a relational database from scratch is cumbersome. One has to write down the UML, pass the schema to the database, and eventually write code to access the database. This initial effort is a bit high, but one could say it only happens once. But perhaps not quite. Considering industry applications, requisites will evolve overtime, migrations are also going to be needed.

Could these barriers be lowered and the process do have a complete relation database working be quicker?

Context

Unless the project is of very high simplicity requiring only several static webpages, it is most likely to require a database backend.

Solution

Considering the importance of databases in these applications, Web Application Frameworks pay the utmost attention to usability and abstract their use. For example, patterns such as Active Record (section 2.2.3) and Data Access Object (section 2.2.2) are commonly employed.

Having to write Active Record pattern by hand would almost defeat the point. Web Frameworks that use this pattern\(^2\), will automatically generate code to quicken the development process. Properties of model's class are the same as the database schema — this is called Object-Relation

---

\(^2\)Ruby on Rails for example.
Mapping (ORM) and can be used to reduce the amount of code and time needed to handle a database. Additionally, they might provide database migration support. The level of support varies and a completely automated conversion from one schema version to the other is typically not found.

### 2.3.2 Model Generation

#### Problem

Let us ponder for a moment what creating a new model is. At first, the most obvious answer will be: it is just a matter of creating a new class. Is it?... Where to add it? Which class does it extend? What methods should already be implemented? How is the database changed? Since migrations are going to be needed, how can we perform them? Questions like these are very important for a developer who is inexperienced with a given framework. The problem then, is not so much creating the model, rather creating the model correctly and dealing with effects this change will cause.

#### Context

Unless it is a static website without requiring any database, models are most likely going to be required. From the perspective of a Web Application Framework, creating a new model is a subtle way to indicate the programmer should be clearly separating his data models from his business logic. Subtly enforcing the MVC design pattern, even if the developer is unaware of this.

#### Solution

The framework should provide a command that creates a new model with a given name and a set of attributes. At this time, the framework could perform additional operations. These operations might involve updating the database schema, generate new code for Active Record or Data Access Object patterns, possibly updating needed configuration files. The command should be self-sufficient, that is, the project should be able to compile and run without additional effort from the programmer.

### 2.3.3 Scaffolding

#### Problem

While we can argue that when applying the MVC pattern, creating a model is the simplest part. Now controllers and views are required to manipulate this model and to make it useful. Implementing the Create, Read, Update and Delete (CRUD) operations for a model will certainly take plenty of the developer's time. Can a sufficiently good solution be generated for any model with any set of attributes?
Background

Context

Let’s consider the project already has a few models, now the next logical step is to add views and controllers to manipulate them. The CRUD operations should fit most cases, and perhaps the developer is just seeking to have a working prototype quickly to demo the project to his client or stakeholder.

Solution

The framework should employ a technique known as scaffolding. While templating\(^3\) is applied in a on-demand basis, scaffolding is generated code requested by the programmer. In the context of web frameworks, the code they generate are generally a Model, a View or a Controller. Scaffolding then, can be used for example to generate models from database or views from the model.

Scaffolding quickens implementation of the CRUD operations, but further functionality is up to the developer. Sometimes, however, for rapid prototyping\(^4\) this might prove to be enough [Dav06].

2.3.4 Continuous building

Problem

Manually building a project is not an issue until a developer has experienced how the workflow can be improved using automated builds. The issue of having multiple builds and testing also make continuous building attractive.

Context

The web application the programmer is working on is required to have responsive web design\(^5\). The current setup involves testing on a few devices which need to be refreshed everytime the page changes.

Solution

The framework should have a build tool, or take advantage of an existing one. This build tool should be able to detect code changes and rebuild project instantly, providing immediate feedback to the developer. This build should be incremental so the total time does not exceed a few seconds. Moreover, the build tool could also be running all the existing tests. If possible and if the number of tests is very high or slow, they should be ordered. That is, the first tests to be executed are the ones that test the code that has been altered since the last run.

---

\(^3\)Typically templating is the process of controller passing the data to a view and applying the needed data bindings.

\(^4\)Rapid prototyping focuses on fully functional prototype, which consists of “[…] parts that are close to or are in the final shape. […] the time to produce any part […] will be fast and can be in a matter of hours.” [Chu10]

\(^5\)An approach aimed at crafting sites with an optimal viewing experience across all devices shapes and forms.
2.4 Android

So far we have only discussed a theoretical overview of Scala functional programming and Web patterns. But how do they bind together, in an Android development environment?

When looking at database patterns (presented in sections 2.2.2, 2.2.3, 2.3.1) we can see that all of them could be applied to Android. However, the current support within Android API and IDE is just a simple inclusion of SQLite. Continuing to analyse other patterns, at the moment of writing, Android Studio does not support any scaffolding techniques. There is definitely room for improvement in this area, and it is possible to build upon the same principles web frameworks have.

These patterns have real value in Web Application Frameworks and have a positive impact in the development workflow. To a certain extent, this means that Web Development can ease Rapid Prototyping [Bol11]. As such, the gap between Android and Web Frameworks should be reduced.

For a successful implementation of the patterns presented in this chapter, a broad State of the Art was performed, analysing build tools and other tools, Web Application Frameworks and Android development itself. The next chapter will present the results of such investigation and evaluate how they could be part of the framework.
Background
Chapter 3

State of the Art

This chapter discusses existing work and research relevant to the context of the problem.

In section 3.1, we start by reviewing Android itself to understand what existing technologies are used and what limitations they have. In section 3.2 we examine several Web Frameworks from different programming languages to provide a solid background on their workflow, patterns and methodologies and put them in perspective against Android. Then, in sections 3.3 and 3.4, we look at existing build tools and plugins that allow the use of Scala in Android applications.

In the end (section 3.5), a global overview of state of art is made and some conclusions are drawn.

3.1 Android

Android is an operating system from Google designed primarily for touchscreen mobile devices. As mentioned in the Introduction’s Context section 1.1, it currently has the highest smartphone marketshare worldwide, nearing the 80% mark; [Gar, Str] and it has over 1 million applications officially published [Gsm].

Android uses an optimized JVM called Dalvik Virtual Machine [Ehr10]. It uses less memory and its design allows multiple instances of JVM to run simultaneously more efficiently. Even though it uses a different Virtual Machine, it is as capable to run Scala as it is Java, since both languages compile to the same Java-bytecode. ART\(^1\) is a new Android runtime currently being developed, that will possibly be included in next release of Android [Gooc]. It uses ahead-of-time (AOT) compilation which eliminates the need to use a Virtual Machine to interpret an application — this translates into faster application start up and execution times\(^2\) [SML]. There are two


\(^{2}\)For example, a comparison between both runtimes showing that ART is consistently faster than Dalvik running an implementation of quicksort algorithm can be found here: https://docs.google.com/spreadsheet/ccc?key=0AnE-DX9Fbr9KdFZtR1hQaWxMG5XzDR3TR1Q3JONm.
important trade-offs: the code will occupy more memory and the installation will take more time. The former problem has to do with the fact that native code is itself less dense than bytecodes\(^3\) [SML]. The latter problem has to do with the fact that ART’s AOT compilation will be done only at the device that is installing the application, in order to be optimized for its hardware. As of time of writing, there no empirical study has been conducted to assess how Scala performance changes. It is the author belief that, as it gets faster for Java, the same will be true for Scala, albeit not necessarily in a 1:1 relation.

The development in Android is not bound to any particular operating system, computer architecture, IDE or language. In the next section 3.1.1, a feature analysis of Android Studio IDE is made.

### 3.1.1 Android Studio IDE

Android Studio is the official next-generation IDE for Android application development. As of June 2014, it is still considered early access preview [Good]. It will eventually replace the Eclipse plugin solution. It was based on the IntelliJ IDEA Community Edition\(^4\) [Goob].

When creating a new project, there are 3 available templates to start from:

- **Blank Activity** — Base project structure, with an empty activity\(^5\) and view\(^6\).

- **Fullscreen Activity** — Base project structure, simple view layout and simple activity with some interaction - changing visibility of the Android UI system bar.

- **Master/Detail flow** — Base project structure, 2 layouts: one tablets or wide-screen devices and one for smartphones and their respective views.

After the project is made, there are more templates for new activities:

- **Login Activity** — Generates a layout with username and password fields, and an activity that already contains actions and an AsyncTask\(^7\) partially completed — notably missing the network connectivity to actually validate the login.

- **Settings Activity** — Generates an activity with some interaction with the Android’s application SharedPreferences.

Android has a XML file called AndroidManifest.xml — which is used to describe the application and what components — activities, permissions, services, etcetera — are being supplied by that application [Mur08].

---

3 Bytecode are instructions that will be interpreted by a Virtual Machine and not directly by the CPU.

4 More information can be found on [http://www.jetbrains.com/idea/](http://www.jetbrains.com/idea/).

5 Activities are the controllers that handle the views content and actions. More information is available on [http://developer.android.com/reference/android/app/Activity.html](http://developer.android.com/reference/android/app/Activity.html).


7 AsyncTask is a thread handled by Android to abstract common tasks such as networking, that require background processing. More information is available on [http://developer.android.com/reference/android/os/AsyncTask.html](http://developer.android.com/reference/android/os/AsyncTask.html).
State of the Art

There are additional blank templates for each Android components (Blank App Widget, Blank Fragment, Broadcast Receiver, Content Provider, Custom View, Daydream, Intent Service, List Fragment, Service) which alter the manifest file accordingly.

It is important to note that since a Login activity typically requires networking access, a permission to allow an application to use this resource is needed; however, the template provided from Android Studio IDE does not automatically add this permission to the application’s manifest, relying on the user to manually add it later.

A study analysed the permissions request patterns on applications and found that "one-third" were overprivileged [FCH11], concluding that:

“Results show that applications generally are overprivileged by only a few permissions, and many extra permissions can be attributed to developer confusion. This indicates that developers attempt to obtain least privilege for their applications but fall short due to API documentation errors and lack of developer understanding.” [FCH11]

3.1.2 Android Developer Tools

The Android Developer Tools is an official plugin for Eclipse to provide a development environment for building Android applications [Gooa]. However, as mentioned in previous section 3.1.1, it will soon be replaced, as Google is pushing developers to use their new Android Studio IDE.

In terms of features, it is similar to Android Studio IDE. The same templates are present, however the build system is much more limited [Wha12]. Currently, the external dependencies must be handled by adding JAR files manually which can pose problems for medium to large scale application development [Wha12].

Writing tests for an application is possible but it requires a new and separate project specifically for testing.

3.1.3 Scaloid

Scaloid is a library for Android to abstract its API by leveraging Scala’s language features. It is under active development, and as of 1st of June 2014 it has reached the version 3.411.

It contains several noteworthy features:

• UI Layout without XML — Android development kit promotes XML to build user interface layouts. However, XML can be quite verbose and is not type-safe. Scaloid composes the user interface layout in Scala, therefore achieving both clarity and type-safeness. The downside of using code to program layouts is related to different screen sizes and orientations — normally, there would be a XML stored in different folders named after the screen

---

9 Eclipse is an IDE; more information can be found on https://www.eclipse.org.
10 For more information, please consult: http://developer.android.com/tools/testing/testing_eclipse.html.
11 More details about this version are available on blog.scaloid.org/2014/05/scaloid-34-is-released.html.
State of the Art

density and orientation; then, Android would load the correct one automatically. To solve this problem using Scaloid, the programmer has to write additionally code check screen size and orientation to apply the correct layout.

- **Implicit conversions** — Scaloid employs several implicit conversions\(^\text{12}\). They are used for example to convert a `String` into a `URI`\(^\text{13}\).

- **Traits** — Scaloid contains several traits\(^\text{14}\) to abstract several aspects in Android programming. These traits provide simple functionality and in general, make API calls shorter and more expressive. As an example, listings 3.1 and 3.2 show a use case where a possible trait can be applied.

```java
1 startActivity(new Intent(context, classOf[MyActivity]))
```

Listing 3.1: Starting an activity without using Scaloid traits

```java
1 startActivity[MyActivity]
```

Listing 3.2: Starting an activity using Scaloid traits

For exemplification purposes, consider the following Android code, without the use of Scaloid library, as shown in listing 3.3.

```java
1 val button = new Button(context)
2 button.setText("Greetings")
3 button.setOnClickListener(new OnClickListener() {
4   def onClick(v: View) {
5     Toast.makeText(context, "Hello, you just pressed a button!", Toast.LENGTH_SHORT ).show()
6   }
7 })
8 layout.addView(button)
```

Listing 3.3: Android code without using Scaloid library

The equivalent code is reduced to one line, as shown in listing 3.4.

```java
1 SButton("Greetings", toast("Hello, you just pressed a button!")
```

Listing 3.4: Android equivalent code using Scaloid library

\(^\text{12}\)Implicit conversions is a Scala feature; more information available on [http://docs.scala-lang.org/overviews/core/implicit-classes.html](http://docs.scala-lang.org/overviews/core/implicit-classes.html).

\(^\text{13}\)Uniform Resource Identifier, it serves to identify a web resource.

\(^\text{14}\)Traits is a Scala feature; more information available on [http://www.scala-lang.org/old/node/126](http://www.scala-lang.org/old/node/126).
State of the Art

Naturally keeping up with the official API is an ongoing effort — at the moment the supported API is at the level 8 or Android 2.2 version\textsuperscript{15}. Nevertheless if some feature is not present in Scaloid, it is still possible to use the Android API without much effort.

3.1.4 AppDF

The App Description File (AppDF)\textsuperscript{16} is a format that aims at full interoperability among Android applications stores, when submitting an application. There are plenty of alternative stores besides the official Google Play Store [App13] and not considering them will limit the number of potential users.

This solution mitigates the problem of dealing with every submission form idiosyncrasies, and makes it a matter of just uploading one file.

3.2 Web Application Frameworks

In the following sections (3.2.1, 3.2.2, 3.2.3), three known and stable Web application frameworks from different programming languages are reviewed. For each, methodologies and patterns will be discussed as well as console commands examples for each of the following tasks: project generation, compilation, testing and scaffolding.

In section 3.5 we are going to compare them in terms of features.

3.2.1 Play!

Play Framework\textsuperscript{17} is a web application framework that runs in the JVM with a very capable modular architecture that makes it straightforward to write plugins [Pla]. Scala and Java languages are both fully supported, each have their own API bindings to respect the languages nomenclature conventions.

The framework is RESTful\textsuperscript{18}, which promises a much higher reliability. Furthermore, it is capable of highly-scalable applications as it consumes "predictable and minimal resources from CPU, memory and threads" [Pla].

\textit{Command-line utility}

The framework generates the files structure automatically when a new project is made, with basic MVC folder organization, with the command instruction as shown in listing 4.1.

\footnotesize{\textsuperscript{15}As of June 2014, the current Android API level is 19 or Android 4.4 version
\textsuperscript{16}Project is being hosted at: \url{https://github.com/onepf/AppDF}.
\textsuperscript{17}For more information consult \url{http://www.playframework.com}.
\textsuperscript{18}REPresentational State Transfer, \url{http://www.oracle.com/technetwork/articles/javase/index-137171.html}.}
At this stage, the developer has to choose between Scala or Java.

Listing 3.6 demonstrates how to run a project. The project is compiled and the default configuration launches a simple local server.

```
1 play run
```

Listing 3.6: Compiling and running the project

It is equally simple to run the tests of the project, as listing 3.7 demonstrates.

```
1 play test
```

Listing 3.7: Running all tests in the project

It is interesting to note that Play framework itself uses SBT extensively. In fact, Play is just a plugin for SBT. This approach has several advantages. For example, continuous tasks (that are triggered every time the code changes) can be made just by adding the character ~ to any command Play has. Another example of this approach advantage is that updating to a newer version is just a matter of refreshing the project dependency.

### 3.2.2 Ruby on Rails

Ruby on Rails\(^{19}\) was created in 2003 and it is an open source web application framework which runs on the Ruby programming language [oR].

The framework promotes the use of known software engineering patterns such as Convention over Configuration\(^{20}\), active record pattern\(^{21}\) and MVC.

Ruby on Rails includes tools that make common development tasks easier out of the box, such as scaffolding, a simple Ruby web server, and Rake which is a build system — this makes the framework incorporate a fully functional basic development environment [oR].

#### Command-line utility

Like the Play! framework, Ruby on Rails also generates the file structure automatically when a new project is made, with basic MVC folder organization, by using the command instruction as shown below in listing 3.8.

---

\(^{19}\)For more information consult http://rubyonrails.org.

\(^{20}\)This essentially means frameworks should assume reasonable defaults, without requiring unnecessary configuration effort.

\(^{21}\)This pattern describes a domain object that knows how to interact directly with database tables [Fow02].
State of the Art

Listing 3.8: Ruby on rails — creating a new project from scratch

```
rails new simpleRailsProject
```

Listing 3.9 demonstrates how to run a project. The project is compiled and the default configuration launches a simple local server.

```
rails server
```

Listing 3.9: Compiling and running the project

It is equally simple to run the tests of the project, as listing 3.10 demonstrates.

```
rake test
```

Listing 3.10: Running all tests in the project

Ruby on Rails scaffolding capabilities are demonstrated in listing 3.11. The first command will generate a controller named `welcome` and view `index`; the second one will generate a model named `Post` with title and text attributes already defined, along with the appropriate database support.

```
rails generate controller welcome index
rails generate model Post title:string text:text
```

Listing 3.11: Compiling and running the project

3.2.3 Django

Django\(^{22}\) was created in 2005 and is an open source web application framework that was written in Python programming language. It encourages "rapid development" by using pluggable reusable components and it follows the MVC architectural software design pattern [Fou].

Django also provides an optional administrative CRUD operations interface that is generated dynamically through introspection and configured via admin models [Fou].

Command-line utility

Like the other frameworks previously described, Django generates the file structure automatically when a new project is made, with basic MVC folder organization, by using the command instruction as shown below in listing 3.8.

```
django-admin.py startproject simpleDjangoProject
```

Listing 3.12: Ruby on rails — creating a new project from scratch

\(^{22}\)For more information consult [https://www.djangoproject.com](https://www.djangoproject.com).
State of the Art

Listing 3.13 demonstrates how to run a project in Django. The project is compiled and a simple local server in launched on port 8000.

Listing 3.13: Compiling and running the project

1 manage.py runserver 8000

The command shown in listing 3.14 demonstrates how to run all the tests of a project.

Listing 3.14: Running all tests in the project

1 manage.py test

When it comes to scaffolding, Django does not use a console command. Instead, it provides an administration interface that ultimately has the same result.

3.3 Build tools

Build automation can be described as automating tasks that are needed for software developers to handle compilation, packaging, static code quality analysis, testing, deployment, creating documentation or release notes, among others. This automation is typically handled by a build tool. This project’s goals are related to this and therefore it makes sense to analyse current build tools that could potentially be adopted or extended to meet the objectives.

3.3.1 Maven

Maven is a project management framework\(^{23}\), or to put it simply, a build tool. It is used mostly for Java projects but it also copes with Scala and other programming languages. The most relevant features for this thesis are as follows:

- Simple project setup that follows best practices
- Extensible via plugins in Java or in other scripting languages
- Dependency management

A Maven project configuration is stored in a single XML, which can be quite verbose. Listing 3.15 is an example for a simple project dependency.

Listing 3.15: Example for a simple project dependency

1 <dependency>
2   <groupId>junit</groupId>
3   <artifactId>junit</artifactId>
4   <version>3.8.1</version>
5   <scope>test</scope>

While Maven’s dependency management is useful, it requires the JAR files to be publicly available in Maven repositories. This can be troublesome for dependencies that are not present in there or are private, requiring an private Maven repository [YL12].

### 3.3.2 Simple Build Tool

Simple Build Tool\(^{24}\) (SBT) is a build tool for Scala and Java projects.

The most relevant features for this thesis are as follows:

- Scala-based build definition that can use the full flexibility of Scala code
- Continuous compilation and testing with triggered execution
- External project support (git repository can work as a dependency)
- Extensible via plugins

This set of features are very useful and important as they allow the creation of a new plugins to define new build tasks in Scala while also getting the continuous tasks for free.

If we consider the open source Scala projects hosted at GitHub\(^ {25}\) we can see it is the most popular build tool [Giti]. Admittedly, this observation alone does not represent the whole, it shows a generalized preference towards SBT.

It is relevant to note that SBT uses Maven Central repository\(^ {26}\) as the default dependency management resolver for external project dependencies, allowing them to be used in Scala projects.

### 3.4 Existing SBT Plugins

In this section, plugins for SBT are presented. With the use of these, actual development of a Scala Android application is possible. The projects are open-source and they are hosted at GitHub.

Since we did not plan to reinvent the wheel — we *could* end up with something not as rounded shaped — we could use one of these projects as an external dependency and contributions could be made directly to these projects.

#### 3.4.1 jberkel/android-plugin

This project started in 2009 and has been maintained consistently having over 20 different people contributing to its development. The plugin provides the following features: managing emulator, building, deploying and testing.\(^ {27}\)
State of the Art

Syntax is not the most simple to use; an example of a emulator test command shown in the listing 3.16.

```bash
```

Listing 3.16: Example of SBT jberkel/android-plugin command

To use this plugin, it is required to add it as an external dependency to an existing project, or alternatively, use a template Android project that already has included it.

Being the oldest project for Android with Scala, it widely used and has had a numerous contribution over time, so it is probably the most stable. Unfortunately it also meant the main contributor has his lost interest and the project hasn’t been updated much in the last year.

**As a support for proposed solution**
This project in particular can be used as the base to solve the problems related with building an Android project.

### 3.4.2 pfni/android-sdk-plugin

This plugin is more recent when compared with the others, and was created with the ease of configuration as its primary objective.\(^{28}\)

The structure it generates is different from the default SBT and more similar do what Eclipse generates — in a way, it looks like a normal Android project.

**As a support for proposed solution**
This project can also be used as the base to solve the problems related with building an Android project.

### 3.4.3 scalastyle/scalastyle-sbt-plugin

Analysing computer software without actually executing programs is called **static program analysis** [Wic95]. The analysis is usually done by an automated tool which usually outputs a document that contains the code review, for programmers to inspect and make the necessary changes if needed.\(^{29}\)

This particular open-source project performs static Scala code analysis and is already encapsulated in a SBT plugin. The output is compatible CheckStyle\(^{30}\) which has a plugin for Eclipse for easy visualisation of possible code faults or improvement suggestions.

**As a support for proposed solution**

---

\(^{28}\)Project is being hosted at: [https://github.com/jberkel/android-plugin](https://github.com/jberkel/android-plugin).

\(^{29}\)Project is being hosted at: [https://github.com/scalastyle/scalastyle-sbt-plugin](https://github.com/scalastyle/scalastyle-sbt-plugin).

\(^{30}\)More on this project at: [http://checkstyle.sourceforge.net](http://checkstyle.sourceforge.net).
This project performs static code analysis and could be an useful inclusion to the solution. Since this project is already a plugin for SBT, the integration should be simple and direct.

### 3.4.4 mpeltonen/sbt-idea

This project creates IntelliJ Idea\(^{31}\) projects files. After generating the appropriate files it is possible to open the project in the IDE.\(^{32}\)

**As a support for proposed solution**

Generating the necessary files in order to a project compatible to a specific IDE is the next logical step to take towards a better solution. This plugin is then considered useful and should be included.

### 3.4.5 typesafehub/sbteclipse

This project creates Eclipse\(^{33}\) projects files. After generating the appropriate files it is possible to open the project in the IDE.\(^{34}\)

**As a support for proposed solution**

The reasons mentioned in previous section 3.4.4, still hold valid for this plugin and it should be included also.

### 3.4.6 olim7t/sbt-scalariform

This project formats the projects Scala source files using Scaliform\(^{35}\), a code formatter.\(^{36}\)

**As a support for proposed solution**

This project provides Scaliform integration with SBT. In pratice, this integration allows the projects source code to be completely formatted whenever it is compiled, ensuring it follows the predefined set of rules for proper indentation across all files. This project then, could lead to projects better formatted overall, potentially increasing slightly its readability and maintainability.

### 3.5 Conclusions

Table 3.1 compares the Android Studio IDE presented in section 3.1.1 and the web frameworks analysed (sections 3.2.1, 3.2.2, 3.2.3) to the proposed solution, in terms of features.

All the mentioned frameworks meet the demands and expectations of a software engineer, providing a series of techniques, methodologies, patterns and self-contained tools to accomplish

---

\(^{31}\)This is known an IDE, more infomation here: http://www.jetbrains.com/idea

\(^{32}\)Project is being hosted at: https://github.com/mpeltonen/sbt-idea.

\(^{33}\)This is known an IDE, more infomation here: http://www.eclipse.org

\(^{34}\)Project is being hosted at: https://github.com/typesafehub/sbteclipse.

\(^{35}\)Project is being hosted at: https://github.com/mdr/scalariform.

\(^{36}\)Project is being hosted at: https://github.com/olim7t/sbt-scalariform.
State of the Art

Table 3.1: Feature analysis comparison between the several technologies

<table>
<thead>
<tr>
<th>Feature</th>
<th>Play</th>
<th>Rails</th>
<th>Django</th>
<th>Android Studio</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate project from scratch</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>generate tests structure for tdd</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>generate model</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>generate controller</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>generate view</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>build</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>test</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>continuous build</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>continuous testing</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

their tasks in a simple manner. Android Studio IDE is the next-generation tool from Google, but it still lacks certain capabilities found on common web frameworks, notably scaffolding — there are several templates present to quicken the start of the writing of a new component, but these are not as powerful or complete.

As presented in this chapter, there are several projects that, if tightly integrated, could allow the creation of a solution that succeeds in all the features presented in the previous table. This will be explored in the next chapter where the implementation of such solution is detailed.
Chapter 4

Rapid Prototyping and Development Framework

4.1 Overview

The framework was implemented as a Simple Build Tool (SBT) plugin. SBT is a build tool for Scala and Java projects and is currently considered the de facto build tool for Scala projects. The decision to develop this project as a SBT plugin has other several inherent advantages: (i) other existing SBT plugins can be used to create a more complete development environment, (ii) Android building operations are already solved by one plugin, (iii) we don’t introduce accidental complexity deriving from learning and using an entirely new tool. This plugin, agile-scala-android, is open-source and free to use.

When designing a framework to handle and improve the process of software development, a valid approach can make the use of multiple external dependencies to fill-in the gaps and to speed up development. By tightly integrating existing tools or plugins, we can avoid re-inventing the wheel, and make the whole experience more fluid. In practice, it also means the development of other tools can evolve independently and move at their own pace.

The framework makes use of metaprogramming, more specifically template metaprogramming. This is code that is generated when the project is going to be compiled, using Reflection features of the Java Virtual Machine, leaving the possibility for added functionality that the developer did not write explicitly, without the runtime performance impact usually associated to other reflection techniques.

Figure 4.1 illustrates the external plugin dependencies of agile-scala-android. The Android SDK Plugin handles the build process. Scalariform reformats code to a specific guide-line. Scalastyle is a static code analysis tool that checks for possible problems in the code, mainly formatting issues. Additionally, there are two plugins which convert the current project to IntelliJ and Eclipse

\footnote{Complexity which is non-essential to the problem being solved.}
\footnote{The project is available at: https://github.com/luismfonseca/agile-scala-android.}
\footnote{Also preventing the possibility of ending up with a square wheel.}
Rapid Prototyping and Development Framework

Figure 4.1: Architecture overview with external plugin dependencies.

--- these generate the necessary project metadata in order to be correctly interpreted by these IDEs. Finally, the developed features of our framework do not overlap the goals of these plugins, they complement them. These features are better explained in the next section 4.2.

4.2 Framework Functionalities

We now proceed to analyse the implemented features. Each block is discussed considering (i) motivation, (ii) implemented solution and (iii) impact in development workflow.

4.2.1 New Project

Create a new empty Hello World Android project with Scala from the scratch.

Motivation — Currently, starting a new Android project with Scala is not a trivial task. The first problem to solve is choosing the Android SBT Plugin to build the project, and there are only two valid options:

(a) Android Plugin by jberkel\(^4\) — The recommended approach to start a new project with this plugin is to use giter8\(^5\) and have the project generated from a simple template. Naturally, the giter8 tool needs to be installed first and then the template to be used needs to be updated. As of this writing, the officially recommended template has not been updated since May 31

\(^4\)Project is being hosted at: https://github.com/jberkel/android-plugin.

\(^5\)This is a command line tool that applies templates hosted on Github. The project is being hosted at: https://github.com/n8han/giter8.
of 2013 (over a year ago). In short, this option requires additional effort to install other tools and the project will be based on an outdated template.

(b) Android SDK Plugin by pfn\(^6\) — The recommended approach to start a new project with this plugin is to use Android Studio or the Android SDK tools. This effectively overcomes the problem of having a outdated version of the template. However, after this step, the developer is expected to create three configuration files in order to allow SBT to use this plugin and import its build definitions.

If we look at the web framework Play!\(^7\), creating a project is simply just a one line command, as shown in Listing 4.1.

```
1 play new simplePlayProject
```

Listing 4.1: Creating a new project from scratch using Play! Framework

**Implemented Solution** — The approach taken was inspired in simplicity of np SBT plugin\(^8\). This plugin can create a SBT project anywhere, generating a proper structure and configuration files that are ready to be extended. It is minimalistic in the sense it generates only the bare minimum to get a project set up and have the developer start coding right away.

Combining this concrete implementation and what web frameworks offer, we created the npa\(^9\) command which accepts just two arguments: (i) default package name (i.e. pt.pimentelfonseca.example) and the Android SDK API version (i.e. 19 which refers to Android 4.4).

---

\(^6\)Project is being hosted at: https://github.com/pfn/android-sdk-plugin.

\(^7\)For more information consult http://www.playframework.com.

\(^8\)np is short for New Project. This plugin is being hosted at: https://github.com/softprops/np.

\(^9\)npa is short of New Project Android.
Figure 4.2 illustrates the events that occur when the command is executed. The necessary folder structure is created, along with the Android files needed for a minimal Hello World application; additionally, a *gitignore* file is created, allowing Git to ignore common artefacts that are temporarily generated when building an Android application. Finally, SBT configuration files are generated, importing and configuring the required plugins and libraries for the project. As this step changes SBT’s currently loaded build definitions, the developer is warned that a *reload* command needs to be executed immediately afterwards.

The libraries included by default support JSON parsing, Slick Database and Android SQLite JDBC interface.

**Workflow** — The steps necessary to create a new Android project with Scala are as presented in Listing 4.6.

```
$ mkdir simple
$ cd simple
$ sbt
> npa pt.simple.project
> reload
```

Listing 4.2: Creating a new project using *agile-scala-android*

After these commands are executed, the developer is ready to extend and edit the application to meet his requirements, possibly taking advantage of the Model generation (section 4.2.2) and Scaffold (section 4.2.3) features next.

### 4.2.2 Model generation

Generates a new Model with a given name and a set of attributes.

**Motivation** — Creating a new Model is usually a very basic task. Simply put, one needs only to create a new Class with a set of attributes that represent and encapsulate the abstraction over a given domain. This feature on its own is therefore very elementary. However, it must not be viewed purely as an independent piece. Having this notion of Model subtly indicates the programmer that he should be separating clearly his data Classes from view and business logic. So even if unknowingly, the programmer will start following the MVC pattern. Additionally, it also works to create a more fluid workflow as the programmer is likely to perform other operations through the command line utility.

Given these advantages, this command can also be found in some web applications (i.e. Ruby on Rails).

**Implemented Solution** — Models being purely an algebraic type are best expressed via Scala’s Case Classes.

---

10. These are image files, layouts and menu files, resources files, and a manifest file.
11. Git is a popular version control system and source code management.
Rapid Prototyping and Development Framework

Figure 4.3: Generate a new Model sequence diagram.

The Model name should be written in English and in singular form as this will be important for the Scaffold and Database operations. When a model is generated, it will be placed inside the models package that belongs to the project’s default package — this will help in terms of code structuring.

As depicted in Figure 4.3, besides the name of the Model, the developer should specify a number of fields right away, but these are not final and can be extended later on. Given a set of fields, the supporting filed types are automatically resolved. For instance, *Date* requires an import, in this case `import java.util.Date`.

The list of currently supported types are as follows: Char, Int, Long, Float, Double, Boolean, String, Array and Date. Additionally, other Models can be used as field types.

**Workflow** — At this point, the programmer has most likely just finished creating a brand new project and is seeking to add Models to start of the development of the application. Alternatively, the application is already in a functional state but there were more requirements that led to new Models being necessary. Let’s assume that the former is the case and the project is to create a typical Blog application. Listing 4.3 describes a possible set of commands to create three models: Author, Comment and Post.

```scala
1 $ sbt
2 > generate Author name:String
3 > generate Comment text:String author:Author
4 > generate Post title:String text:String views:Int author:Author publishedDate:Date comments:Array[Comment]
```

Listing 4.3: Generating several new Models using *agile-scala-android*

The result will be three case classes being generated, with the needed imports resolved. Figure 4.4 illustrates in UML the model schema generated.
The developers can now continue modelling the domain of their application or use the Scaffold feature next.

4.2.3 Scaffold

Automatically generate the Views and Controllers necessary to implement the CRUD operations on a given Model.

Motivation — In the context of web frameworks, scaffolding is the process of generating working code for a View or a Controller. It can be used, for example to generate models from database or views from the model. In contrast, web frameworks also have templating — but this is a View that is applied in a on-demand basis, while scaffolding is code generated on request by the programmer.

This code generation solves a very important aspect of programming projects in general\textsuperscript{13} — starting a project is a hard step, specifically if there are relatively unfamiliar technologies involved. Having the CRUD operations on the models instantly can help to have an immediate visual feedback and provide an implementation that is valid for prototyping and that can be later on extended to meet the project’s requirements; a starting point.

\textsuperscript{13}This also applies to non-programming projects as well.
Rapid Prototyping and Development Framework

Figure 4.5: Generate a new Model sequence diagram.

**Implemented Solution** — Since enforcing MVC is partly the goal of the framework, the Models that the plugin will scaffold must be present under the models package. The project needs to be free of errors as it will be compiled. This is a necessary step as the model’s `.class` file will be dynamically loaded using a `ClassLoader` and then the fields will be gathered through Reflection.

The views are generated and composed of small templates that are applied according to their field names and types. However, there are 2 special cases:

(a) The field type is another model. In this case, the scaffold process will repeat for that model. If this model also contains other models, it will go on to scaffold recursively, until every needed Views and Controllers are generated.

(b) The field type is an array. In this case, another set of templates are used. Currently it is limited to arrays of other models. Nonetheless, this should reflect most use-cases.

Finally, the Android Manifest file is updated to contain the newly created activities.

**Workflow** — Continuing the Blog application example given in Listing 4.3, let’s now consider that the developer wants to perform the scaffold operation on model Post, as presented in Listing 4.4.

```
1 $ sbt
2 > scaffold Post
```

Listing 4.4: Performing scaffold operation on Post Model using `agile-scala-android`

Since the model `Post` depends implicitly on the `Author` and `Comment` models, in case they haven’t been already, they will also be scaffolded. At this point, the programmer will most likely change the Manifest so that the application launches the Activity that lists the `Posts`. Doing so will result in a complete CRUD application for those models. At this stage, the database backend

---

14There are some special cases being considered, for example if the field name contains `password` and its type is a `String` then the edit field will have the password appearance.
support is missing — this needs to be implemented explicitly by the developer; to this end, section 4.2.4 will explain what support agile-scala-android provides.

Figure 4.6: Screenshots of the resulting application.

Figure 4.6 shows two screenshots taken from the scaffolded application which was completely generated, without any coding.

4.2.4 Database support

Generates SQL tables from existing Models to serve as a database support.

Motivation — A database support is essential to almost every single application of web development, discarding static pages. Even in those cases, a minimalistic Content Management System (CMS) is likely to be a requirement. Its no surprise then that most web application frameworks take great care in planning and integrating a database backend with their framework. Some\textsuperscript{15} use Object-Relational Mapping (ORM) which abstracts the database interactions and queries since the objects will be mapped directly to the database.

Android implementation contains a few API classes to allow the use of SQLite, a simple self-contained transactional SQL database engine — which is good for small applications. Nonetheless, using this API is cumbersome and there aren’t currently any tools provided by Google that achieve the level of of simplicity that ORM has.

\textsuperscript{15}For example: Ruby on Rails, Grails, Play! Framework and Laravel.
**Implemented Solution** — The implemented solution uses Slick\(^\text{16}\), a modern database query and access library for Scala. This decision was based on two aspects: (i) Slick takes advantage of Implicits of Scala and results in a syntax which is type safe and very similar to Scala itself, and (ii) Slick is a Functional Relational Mapping that supports SQLite (that means it is compatible with Android). Not being an ORM has several benefits, mostly in terms of performance which is an important aspect in applications. Our goal is then, to generate Slick tables classes from Models and use Android applications folder for local storage.

Figure 4.7 exemplifies the Blog example described in the previous sections and demonstrates the conversion from Models to Tables. As with the Scaffold feature, the first step is to compile the project and access the models and their fields through a `ClassLoader` and `Reflection`. Then, we can examine the fields and temporarily create a database schema that represent the Models and their foreign relationships (these can be 1-1, 1-n and n-n). Once this schema is created, the tables are generated and included to the project. Helper methods are also included to improve navigating through the model relationships — these methods are mostly a filter by id. In this case, `TablePost` will have a function called `author` which will return the Author of the Post.

![Figure 4.7: A diagram exemplifying the process of generating the database conversion.](image)

Given that these tables are automatically generated, the developer should not change them directly, instead, he should use the classes that extend these tables: `Authors` and `Posts`. These work as a Data Access Object (DAO) pattern, separating the database logic and implementation details and the rest of the business logic that would depend on the database.

As a final consideration, every Model will have a corresponding DAO, which is statically accessible through `App.DB`. This will automatically load the database, and apply all the migrations\(^\text{17}\).

\(^{16}\)For more information consult [http://slick.typesafe.com](http://slick.typesafe.com).

\(^{17}\)Automatic generation of migration files has not been implemented and need to be created manually.
Rapid Prototyping and Development Framework

if needed.

**Workflow** — The needed steps to have a working database ready to be used are two: generate models using the feature presented in section 4.2.2 and then, run the commands present in listing 4.5.

```bash
1 $ sbt
2 > migrateDatabase
```

Listing 4.5: Creating a new project using *agile-scala-android*

The developer has now everything ready and set up to use a database engine with his models schema, available in the rest of his application.

### 4.2.5 Automatic permission resolution

Automatically resolves needed Android Permissions\(^\text{18}\) and includes those that are missing from the Manifest file.

**Motivation** — Android contains a very high number of permissions. As of this writing, in the latest version 19 (also known as *KitKat*), there are 145 different documented permissions. It’s an extensive list that changes overtime, and has been historically growing. We have reached a point where it is very difficult to know and keep track of all these permissions. Assuming for the sake of argument that one developer knows every single one, he still needs to be capable of develop the code which requires a permission and recall at that moment that he needs to add it before running the project, resulting in a runtime error and time being wasted.

If an Android developer does not know he needs a special kind of permission, then there are two possible situations:

(a) The application has a runtime error and it warns about the missing permission.

(b) The application has a runtime error and there are no messages about what possibly went wrong. Fortunately, this situation is less common.

Currently, to the best of the author’s knowledge, there is no tool to help the developer with this issue.

**Implemented Solution** — PScout is a tool developed to map Android API functions calls to their required permissions [AZHL12]. Unlike other works before like Stowaway project [FCH11] and [BKLM12] it has the largest coverage, and more importantly the PScout program and source code are freely available. It works by using static code analysis on the entire Android project, leading to a more complete and extensive coverage. Unfortunately, it does not claim to have 100% completeness and soundness [AZHL12].

The implemented solution requires the project to be compiled. To be effective, before the project is handed-over to an Android build plugin since changes in the Manifest file might be

---

\(^{18}\)Android requires applications to explicitly declare needed permissions in order to access protected parts of its API and interact with other applications.
necessary. It then generates a call graph of the application, by using Reflection and analysing the function calls. Required permissions are then collected by comparing these functions to the mapped Android call graph generated by PScout. Lastly, the project’s Manifest is loaded and permissions are compared and missing permissions are added automatically if needed or a warning is issued depending on the project settings.

Workflow — The goal of this feature was to always help when needed, so it is activated by default, checking permissions whenever the user runs the project. By default, missing permissions are automatically added, but this can be downgraded to a warning if the developer prefers.

To manually check the permissions, the developer could run the commands as present in Listing 4.6. This is completely optional as it will always be executed implicitly, so we can argue that the workflow was preserved with this feature.

```
$ sbt
> checkPermissions
```

Listing 4.6: Creating a new project using agile-scala-android

### 4.3 A tutorial approach

Typesafe’s Activator was the chosen approach to create a tutorial for this tool. This is an IDE in a browser and it can be used to bootstrap projects. It has basic IDE features like File management, Syntactical Coloring, Automatic compilation and execution, but more importantly the ability to start a project from a template. There are already numerous templates available, submitted by the community. Usually these templates already have a working environment with the plugins and libraries included and preconfigured. They are also accompanied by a tutorial, which gives a quick presentation of the technologies being used in the template and a set of simple exercises to get the programmer started. This integration with the tutorial is very beneficial as links can be created to open files or execute commands.

The tutorial for agile-scala-android was made publicly available and is being hosted in Github19. Appendix A contains the tutorial at its current state.

### 4.4 Development Effort

The development of agile-scala-android started in February of 2014 as seen in Figure 4.8, and four months later it was deployed and released in the Maven Central Repositories, available freely for anyone to add to their projects. The figure also shows the evolution of the project’s code, measured in code frequency. The last few weeks were more focused in user validation and testing, as will be analysed in section 5.

19The project is available at: https://github.com/luismfonseca/agile-scala-android.
Subjectively speaking, the most challenging features to develop were the Scaffold with recursion and Database backend.

4.5 Conclusions

In this section, a high-level architectural view of agile-scala-android is reviewed and further decomposed each implemented features. These features were analysed not only in terms of implementation, but also in terms of the impact they have on application development workflow. Finally, a general overview of the development effort of agile-scala-android was provided.
Chapter 5

An Academic Quasi-experiment

This chapter details a quasi-experiment within a controlled and synthetic environment using agile-scala-android, aimed to assess the usability and functionality of the developed framework. The developed framework seeks to create a better Android development environment and promote rapid prototyping. However, better can be quite abstract, so in a more pragmatic manner we seek to achieve the following:

- **Development speed** — Improving the overall development speed as this is an essential metric for prototyping.
- **Higher code quality** — This considers the number of bugs per line, cohesion, coupling and execution speed.
- **Easy to use** — Using the plugin should be straightforward, with clean and direct documentation.

To validate these assertions, the experiment approach was defined in section 5.1 and the tasks elaborated (section 5.2). Then, the gathered data is analysed in section 5.4 followed by an objective measurement on the defined metrics in section 5.4. Potential threats to validation are discussed in section 5.5. Lastly, section 5.6 drafts the main conclusions from this experiment.

5.1 Research Design

The study was performed on groups of Software Engineering MSc students which were given the same set of tasks, only divided by two different treatments, viz. (i) the baseline, using Java, and Android Studio or Eclipse and (ii) the experimental, using Scala, and the agile-scala-android. Each group had a total of four participants which were partitioned on the basis of their familiarity with Scala. Pre-test evaluation and post-test questionnaires are used to assess the outcomes of each treatment.
5.1.1 Methodology

We were interested in using experimental approaches for software engineering. Zelkowitz and Wallace grouped twelve experimental approaches into three broad categories [ZW98, Chapter 2]:

- **Observational methods** — “An observational method collects relevant data as a project develops. There is relatively little control over the development process other than through using the new technology that is being studied”.

- **Historical methods** — “A historical method collects data from projects that have already been completed. The data already exists; it is only necessary to analyse what has already been collected”.

- **Controlled methods** — “A controlled method provides multiple instances of an observation for statistical validity of the results. This method is the classical method of experimental design in other scientific disciplines”.

Given our time and human resources, the approach taken was the controlled method, more specifically in a synthetic environment — this type of experiment is composed of a smaller and artificial setting, where the project is not a complete or real product. The main advantages are that (i) the project can be completed within time constrains and (ii) one programmer can perform the tasks alone, while still leading to statistical validity [ZW98].

![Activity diagram for the experiment.](image)

Figure 5.1: Activity diagram for the experiment. The users were divided in to two groups with different treatments. Both had to perform the same tasks and answer the Questionnaire 2, while one had to go through a tutorial before performing the Treatment E.

5.1.2 Treatments

The experiment was composed of two distinct treatments, described as follows:

- **Treatment B, Baseline** — A Java environment was setup. Participants were required to use the IDE they felt most comfortable with, either it being Eclipse IDE or Android Studio.

- **Treatment E, Experimental** — A Scala environment was setup, along with agile-scala-android. Participants were not allowed to use an IDE and had to rely on the use of a command line and Sublime\(^1\) instead, a text editor.

With these two treatments, we can compare how effectively both groups tackled the tasks given and extract conclusions by comparing the results of both.

\(^1\)Sublime is a powerful text editor. For more information consult [http://www.sublimetext.com](http://www.sublimetext.com).
5.1.3 Pre-test Evaluation

Since the experiment was performed on students, to establish the base skill level between the two groups, the average course grade was used. To further complement this information, specific grades on three courses were asked: (i) Algorithms and Data Structures, (ii) Databases, (iii) Object-Oriented Programming. Unfortunately, this latter revealed to be infeasible due to the discrepancy in the course year the participants were.

Table 5.1 presents the participants grades.

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>4</td>
<td>14.350</td>
<td>1.675</td>
<td>0.837</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>15.375</td>
<td>2.641</td>
<td>1.320</td>
</tr>
</tbody>
</table>

Table 5.1: Grouped student current grades statistics

The level of entropy between the grades of the participants is significant but still admissible. A higher number of participating students would most likely lower this deviation and approximate both groups average.

In addition to the objective evaluation using students grades, a background on students previous knowledge took place at this point. Considering this is a self-evaluation, doing this questionnaire before the experiment will prevent the answers from influenced according to one self perception of performance on the given tasks. This background analysis will be detailed further in section 5.3.

5.1.4 Process

This controlled experiment had the objective to assess several distinct claims. The students were split into two different groups, having two different treatments. Each student performed the given tasks alone, using (i) a laptop, (ii) Android tablet Nexus 7 (2014), (iii) internet access, (iv) a paper and (v) a pen.

Data Collection

All students were asked to record the screen while they were performing the experiment. The tablet was not recorded as it is not essential for our data analysis.² From these recordings, we were interested in measuring time and the document the development process in a non-intrusive way.

Tutorial and Questionnaire 1

Before applying the experimental treatment, a tutorial was given to the students (this tutorial can be found in Appendix A). Since they had no prior knowledge of this tool, the tutorial was the first contact and allowed to extract the usefulness and comprehensibility of the developed tutorial. The tutorial contained several exercises, step by step, explaining each feature of the agile-scala-android. Afterwards, a questionnaire was given for feedback.

²Notwithstanding, given that this is a controlled environment, the tablet perhaps could have had the screen recorded as well.
An Academic Quasi-experiment

Tasks

The students were handed a small project for an Android application. This project was divided into several smaller tasks, each having a time limit of 60 minutes — for some tasks in a specific treatment this factor was severely limiting. They were told that the graphic design aspect would not be evaluated, rather completeness and correctness of their implementation. These tasks are looked into with more details in section 5.2.

5.1.5 Post-test Evaluation

At the end of the experiment, a post evaluation questionnaire was handed to the participants. The answers are detailed in section 5.3.

5.1.6 Questionnaires

All questionnaires given out to students were designed using a Likert Scale [Lik32]. This scaling method consists in a measurement of a positive or negative response to a statement. For this experiment, we used five-point Likert items, with the following format: (1) strongly disagree; (2) somewhat disagree; (3) neither agree nor disagree; (4) somewhat agree; (5) strongly agree.

These questionnaires were used to gather the following information:

1. **Background.** Even though in section 5.1.3 an objective view of each group participants background, it is important to assert if the subjective different between the participants regarding basic skills were similar.

2. **Tutorial.** Considering that the tutorial approach was less conventional, this questionnaire was aimed at understanding the value perceived by the participants.

3. **External factors.** It is important to understand if the students felt disturbed or influenced by the fact the screen was being recorded, as this can potentially invalidate the experiment due to the Hawthorne effect [Ada84].

4. **Overall satisfaction.** This group of questions served to judge the subjective self evaluation on how their performance was perceived.

5. **Development process.** These questions were important to consider the problems the subjects had faced when completing the tasks.

6. **Usability satisfaction.** For the participants that used agile-scala-android, a final questionnaire was given to gather additional feedback on the tool.

5.2 Experiment Description

In this section, the experiment tasks are presented as quoted text, and then have a small explained as to why its relevant and the expected results.

---

3Considering the experience on the programming language to be used and the Android development experience.
5.2.1 Preamble

You have been given a job of making a simple note taking application about products. Each Note contains a title, a text, creation date and the Product. Each Product contains a name, a version, and a cost. The user should be able to create, read, update and delete the Notes. The Product however, only needs to have the View implemented. You can use the internet to search for your answers or code. You can also import libraries for your project as you see fit. Keep in mind that each task has a time limit of 60 minutes.

5.2.2 Task 1 — Generate a new project

As your project is just starting, your first task is to create a new project with the following package: pt.fe.up.YOUR_FIRST_LAST_NAME.notex for the Android API version 19. You are free to add dependencies to your project as you see fit.

Description
The very first task was designed to get the project started. The participant has read the introduction, and is ready to start building an Android application. This should not pose any trouble to the participants in either groups.

5.2.3 Task 2 — Implement Models

Implement the models that you consider necessary for this application

Description
The purpose of this task was to get the user to consider the models of the application. Since the complexity of those models is very low, this is a straight forward task. However, there are some considerations regarding (i) the package to hold these models, and (ii) the best field types. The former consideration will have implications later on in Tasks 3 and 4.

5.2.4 Task 3 — Implement Views and Controllers

Implement the CRUD operations needed. Don’t waste time in improving the design of the layouts, keep them functional and simple.

Description
The third task was deliberately designed to be more time consuming and complex if participants were using an IDE. This is because a total of six views add to be constructed along with proper navigation between them. The time limit of sixty minutes was intended to be enforced in this tasks for participants with treatment B, in the best case scenario, one third of the needed views implemented.
5.2.5 Task 4 — Use a database to store Products

Implement a simple database backend. Seed your database with 5 different Products and then use these as the only options for the Notes.

Description
The purpose of this forth task was to evaluate how the databases are implemented and if in this process the models have their fields type simplified to Strings instead of more complex types like Date. After the database module was implemented, a change in the Views also was needed — with this, we can check if Data Access Object pattern was employed or the database was called directly from the Views.

5.2.6 Task 5 — Extra functionality

In this final task, your client requests some extra functionality:

1. Automatically turn on the Wifi whenever a Note is added.
2. Keep the screen awake when viewing a Note.
3. Use what you best see fit to download the following file, and then present it in a Toast message, when opening the application: http://pastebin.com/raw.php?i=1FbCXtkJ

Description
This last tasks is composed of three smaller challenges, each can be solved in approximately in about ten minutes. The interesting observation from this task is that these smaller challenges require permissions to be added explicitly to the Manifest file, otherwise a runtime crash will ensue.

5.3 Data Analysis

We now proceed to analyse the answers gathered the subjects from the questionnaires discussed in section 5.1.6.

Since we used a Likert Scale [Lik32] and the response values are non-linear, we must approach the problem using non-parametric techniques. For this matter, the outcomes of the treatments were compared using the non-parametric, two-sample, rank-sum Wilcoxon-Mann-Whitney test [HWC13], where $n_1 = 4 \land n_2 = 4$. The significance level for all tests was set to 5%. Probability values of $\rho \leq 0.05$ are considered significant, and $\rho \leq 0.01$ considered highly significant.

Let the null hypothesis be denoted as $H_0$, the alternative hypothesis as $H_a$, the baseline group as $G_b$, the experimental group as $G_e$ and $\rho$ as the probability estimator of wrongly rejecting the null hypothesis. There are then three possible alternative hypothesis:

---

4 Assuming that it was not roughly modelled using only Strings.
5 Informally speaking, this test evaluates whether two populations differ with statistical significance, see [HWC13] for details.
An Academic Quasi-experiment

(a) $H_a : G_b \neq G_e$, the baseline group differs from the experimental;
(b) $H_a : G_b < G_e$, the baseline group measurement is lower than the experimental;
(c) $H_a : G_b > G_e$, the baseline group measurement is higher than the experimental.

The corresponding alternative hypothesis are reviewed per question in the sections below.

5.3.1 Background

An objective view on participants background was presented previously in section 5.1.3. Nonetheless, it was important to understand if the basic skills in Android development and programming language was similar across all participants.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Experimental</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bar{x}$</td>
<td>$\sigma$</td>
<td>$\bar{x}$</td>
</tr>
<tr>
<td>B1</td>
<td>3.50</td>
<td>1.29</td>
</tr>
<tr>
<td>B2</td>
<td>2.75</td>
<td>1.71</td>
</tr>
<tr>
<td>B3</td>
<td>4.00</td>
<td>0.82</td>
</tr>
<tr>
<td>B4</td>
<td>1.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 5.2: Succinct overview of the Background results, containing the non-parametric Mann-Whitney-Wilcoxon test.

B1 I have considerable experience with this particular programming language

Let $H_a : G_b \neq G_e$, there was no significant difference ($\rho = 0.3577$) in the scores for the baseline ($\bar{x} = 3.50$, $\sigma = 1.29$) and the experimental ($\bar{x} = 2.75$, $\sigma = 1.71$) conditions, as seen in table 5.2.

B2 I have considerable experience programming for Android

Let $H_a : G_b \neq G_e$, there was no significant difference ($\rho = 0.6060$) in the scores for the baseline ($\bar{x} = 2.75$, $\sigma = 1.71$) and the experimental ($\bar{x} = 2.00$, $\sigma = 1.15$) conditions, as seen in table 5.2.

B3 I have considerable experience with object-oriented programming

Let $H_a : G_b \neq G_e$, there was no significant difference ($\rho = 0.1213$) in the scores for the baseline ($\bar{x} = 4.00$, $\sigma = 0.82$) and the experimental ($\bar{x} = 4.75$, $\sigma = 0.50$) conditions, as seen in table 5.2.

B4 I have considerable experience using build tools

Let $H_a : G_b \neq G_e$, there was no significant difference ($\rho = 0.0631$) in the scores for the baseline ($\bar{x} = 1.00$, $\sigma = 0.00$) and the experimental ($\bar{x} = 2.00$, $\sigma = 1.41$) conditions, as seen in table 5.2.
Given these results, with special attention to questions B1 and B2, we can safely discard different skill levels as a validation threat.

5.3.2 Tutorial

To understand if the tool was perceived as easy to use, a survey was given to participants that were going to use the agile-scala-android, the experimental group. The questions asked are discussed bellow.

<table>
<thead>
<tr>
<th></th>
<th>Experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>x σ</td>
</tr>
<tr>
<td>T1</td>
<td>4.50 0.58</td>
</tr>
<tr>
<td>T2</td>
<td>3.75 1.50</td>
</tr>
<tr>
<td>T3</td>
<td>4.50 0.58</td>
</tr>
<tr>
<td>T4</td>
<td>4.00 1.41</td>
</tr>
<tr>
<td>T5</td>
<td>4.75 0.50</td>
</tr>
<tr>
<td>T6</td>
<td>4.50 0.58</td>
</tr>
</tbody>
</table>

Table 5.3: Succinct overview of the Tutorial results, containing the averages and standard deviations.

T1 The tutorial was considerably easy to understand and follow.

The experimental group gave a score of $\bar{x} = 4.50$, $\sigma = 0.58$, as seen in table 5.3. The responses gathered were positive and very favourable.

T2 I completely understood chapter 1 right away — New (Scala) Project Android.

The experimental group gave a score of $\bar{x} = 3.75$, $\sigma = 1.50$, as seen in table 5.3. The responses gathered were positive and favourable.

T3 I completely understood chapter 2 right away — Creating a new model.

The experimental group gave a score of $\bar{x} = 4.50$, $\sigma = 0.58$, as seen in table 5.3. The responses gathered were positive and very favourable.

T4 I completely understood chapter 3 right away — Database.

The experimental group gave a score of $\bar{x} = 4.00$, $\sigma = 1.41$, as seen in table 5.3. The responses gathered were positive and favourable.

T5 I completely understood chapter 4 right away — Automatic Android Permission resolution.

The experimental group gave a score of $\bar{x} = 4.75$, $\sigma = 0.50$, as seen in table 5.3. The responses gathered were positive and very favourable.
T6 I completely understood chapter 5 right away — Scaffold.

The experimental group gave a score of $\tau = 4.50$, $\sigma = 0.58$, as seen in table 5.3. The responses gathered were positive and very favourable.

Overall, the feedback regarding the tutorial was positive and favourable. There are some areas which were identified as needing some additional work, mainly the New (Scala) Project Android and Database sections.

5.3.3 External factors

We now proceed to analyse the question group regarding external factors. Subjects in experiments can behave differently if they are aware they are being part of an experiment and being recorded. This is known as Hawthorne effect [Ada84]. By considering the answers from participants in question EF1, this validation threat can safely be discarded.

<table>
<thead>
<tr>
<th>Question</th>
<th>Baseline</th>
<th>Experimental</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>EF1</td>
<td>2.75</td>
<td>2.00</td>
<td>$H_a \neq G_e$, $\rho = 0.6017$</td>
</tr>
<tr>
<td>EF2</td>
<td>4.25</td>
<td>4.25</td>
<td>$H_a \neq G_e$, $\rho = 0.8110$</td>
</tr>
</tbody>
</table>

Table 5.4: Succinct overview of the External Factors results, containing the non-parametric Mann-Whitney-Wilcoxon test.

EF1 I felt disturbed and observed by the use of a screencast program.

Let $H_a : G_b \neq G_e$, there was no significant difference ($\rho = 0.6017$) in the scores for the baseline ($\tau = 2.75$, $\sigma = 2.06$) and the experimental ($\tau = 2.00$, $\sigma = 1.41$) conditions, as seen in table 5.4. Overall, both groups gave similar responses, mostly rejecting the fact that their performance was hindered or different due to the fact the screen was being recorded.

EF2 I enjoyed programming in the experiment.

Let $H_a : G_b \neq G_e$, there was no significant difference ($\rho = 0.8110$) in the scores for the baseline ($\tau = 4.25$, $\sigma = 1.50$) and the experimental ($\tau = 4.25$, $\sigma = 0.96$) conditions, as seen in table 5.4. We can conclude that both group of participants enjoyed the experiment.

5.3.4 Overall satisfaction

This group of questions was intended to judge the participants self evaluation on how their performance was perceived.

S1 Overall, this particular setup was suitable for solving every task presented.
An Academic Quasi-experiment

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th></th>
<th>Experimental</th>
<th></th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12345</td>
<td>$\bar{x}$</td>
<td>12345</td>
<td>$\bar{x}$</td>
<td>$H_u$</td>
</tr>
<tr>
<td>S1</td>
<td>3.75</td>
<td>1.26</td>
<td></td>
<td>4.50</td>
<td>0.58</td>
</tr>
<tr>
<td>S2</td>
<td>3.50</td>
<td>1.26</td>
<td></td>
<td>4.50</td>
<td>0.58</td>
</tr>
<tr>
<td>S3</td>
<td>2.50</td>
<td>1.00</td>
<td></td>
<td>4.75</td>
<td>0.50</td>
</tr>
<tr>
<td>S4</td>
<td>1.75</td>
<td>0.96</td>
<td></td>
<td>3.25</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table 5.5: Succinct overview of the Overall Satisfaction results, containing the non-parametric Mann-Whitney-Wilcoxon test.

Let $H_a : G_b \neq G_e$, there was no significant difference ($\rho = 0.1992$) in the scores for the baseline ($\bar{x} = 3.75$, $\sigma = 1.26$) and the experimental ($\bar{x} = 4.50$, $\sigma = 0.58$) conditions, as seen in table 5.5.

S2 I found it very difficult to finish tasks in the given time.

Let $H_a : G_b > G_e$, there was no significant difference ($\rho = 0.1396$) in the scores for the baseline ($\bar{x} = 3.50$, $\sigma = 1.29$) and the experimental ($\bar{x} = 4.50$, $\sigma = 0.58$) conditions, as seen in table 5.5. The results reject the previous hypothesis. This outcome could have been partly influenced by the fact that expectations were too high regarding the total duration of the experiment, leading to some frustration in the end. This observation is disputed in section 5.4 where an objective comparison between the duration of tasks in both groups is made.

S3 I was able to create the user interface at least as quickly as it could normally have been prototyped.

Let $H_a : G_b < G_e$, there was a significant difference ($\rho = 0.0089$) in the scores for the baseline ($\bar{x} = 2.50$, $\sigma = 1.00$) and the experimental ($\bar{x} = 4.75$, $\sigma = 0.50$) conditions, as seen in table 5.5. This confirms the hypothesis that prototyping was perceived to be faster in the experimental group which had access to the Scaffold feature where the baseline group relied on the IDE to visually construct the interfaces.

S4 I found that the resulting application could be used in production-level environments with minimal or no change.

Let $H_a : G_b < G_e$, there was a significant difference ($\rho = 0.0292$) in the scores for the baseline ($\bar{x} = 1.75$, $\sigma = 0.96$) and the experimental ($\bar{x} = 3.25$, $\sigma = 0.96$) conditions, as seen in table 5.5. This confirms the hypothesis that the application can quickly reach a point at which it can be used in production-level environments. It is noteworthy to observe that the baseline group was not favourable with the affirmation while the experimental group was only borderline favourable. Considering these results, it is not safe to conclude that the applications produced by the experimental group are complete and polished, but rather in a more much advanced stage of development.
5.3.5 Development process

The following questions reflected the problems subjects faced when performing the tasks.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th></th>
<th>Experimental</th>
<th></th>
<th>Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12345</td>
<td>(\bar{x})</td>
<td>12345</td>
<td>(\bar{x})</td>
<td>(H_a)</td>
<td>(W)</td>
</tr>
<tr>
<td>DP1</td>
<td>-</td>
<td>3.75</td>
<td>0.50</td>
<td>3.50</td>
<td>1.29</td>
<td>(\neq)</td>
</tr>
<tr>
<td>DP2</td>
<td>-</td>
<td>1.50</td>
<td>0.58</td>
<td>2.50</td>
<td>1.29</td>
<td>(&lt;)</td>
</tr>
<tr>
<td>DP3</td>
<td>-</td>
<td>1.25</td>
<td>0.50</td>
<td>1.25</td>
<td>0.50</td>
<td>(\neq)</td>
</tr>
<tr>
<td>DP4</td>
<td>-</td>
<td>1.75</td>
<td>1.50</td>
<td>2.00</td>
<td>1.41</td>
<td>(\neq)</td>
</tr>
<tr>
<td>DP5</td>
<td>-</td>
<td>3.50</td>
<td>1.00</td>
<td>2.00</td>
<td>1.15</td>
<td>(&gt;)</td>
</tr>
<tr>
<td>DP6</td>
<td>-</td>
<td>3.00</td>
<td>0.82</td>
<td>3.00</td>
<td>2.31</td>
<td>(&gt;)</td>
</tr>
<tr>
<td>DP7</td>
<td>-</td>
<td>3.00</td>
<td>1.83</td>
<td>2.50</td>
<td>1.29</td>
<td>(\neq)</td>
</tr>
<tr>
<td>DP8</td>
<td>-</td>
<td>3.25</td>
<td>1.71</td>
<td>5.00</td>
<td>0.00</td>
<td>(&lt;)</td>
</tr>
</tbody>
</table>

Table 5.6: Succinct overview of the Development Process results, containing the non-parametric Mann-Whitney-Wilcoxon test.

DP1 I found the development style of this setup suitable for a dynamic project

Let \(H_a : G_b \neq G_e\), there was no significant difference (\(\rho = 0.6773\)) in the scores for the baseline (\(\bar{x} = 3.75, \sigma = 0.50\)) and the experimental (\(\bar{x} = 3.50, \sigma = 1.29\)) conditions, as seen in table 5.6.

DP2 I found the development style of this setup suitable for developing face to face with the client

Let \(H_a : G_b < G_e\), there was no significant difference (\(\rho = 0.1396\)) in the scores for the baseline (\(\bar{x} = 1.50, \sigma = 0.58\)) and the experimental (\(\bar{x} = 2.50, \sigma = 1.29\)) conditions, as seen in table 5.6. There is insufficient statistical evidence to support this hypothesis. Nevertheless, the experimental group was slightly more favourable to the idea of developing face to face with the client.

DP3 Concerning this particular setup, most of my difficulties were related with starting a new project

Let \(H_a : G_b \neq G_e\), there was no significant difference (\(\rho = 1.0000\)) in the scores for the baseline (\(\bar{x} = 1.25, \sigma = 0.50\)) and the experimental (\(\bar{x} = 1.25, \sigma = 0.50\)) conditions, as seen in table 5.6. Starting a new project is a simple, straightforward and visual process with an IDE. With this hypothesis we feel that we have enough statistical evidence to claim that using agile-scala-android was just as easy.

DP4 Concerning this particular setup, most of my difficulties were related to gathering needed libraries to support the application
Let $H_a : G_b \neq G_e$, there was no significant difference ($\rho = 0.5866$) in the scores for the baseline ($\bar{x} = 1.75$, $\sigma = 1.50$) and the experimental ($\bar{x} = 2.00$, $\sigma = 1.41$) conditions, as seen in table 5.6.

DP5 **Concerning this particular setup, most of my difficulties were related to building a Graphical User Interface**

Let $H_a : G_b > G_e$, there was a significant difference ($\rho = 0.0375$) in the scores for the baseline ($\bar{x} = 1.75$, $\sigma = 1.50$) and the experimental ($\bar{x} = 2.00$, $\sigma = 1.41$) conditions, as seen in table 5.6. This result confirms that the Scaffold feature is useful for applications that require the implementation of CRUD operations.

DP6 **Concerning this particular setup, most of my difficulties were related to building a Database**

Let $H_a : G_b > G_e$, there was no significant difference ($\rho = 1.0000$) in the scores for the baseline ($\bar{x} = 3.00$, $\sigma = 0.82$) and the experimental ($\bar{x} = 3.00$, $\sigma = 2.31$) conditions, as seen in table 5.6. This result refutes the hypothesis as the baseline group performed better in this test. Reasons for this result might be: (i) Poor documentation of the Slick framework which led to some confusion (when reading the documentation, participants overlooked the much needed package imports and compilation errors were not specific enough) and (ii) the lack of understanding on how the database was being generated on the application. This result means that the documentation provided in the tutorial must be expanded further.

DP7 **Concerning this particular setup, most of my difficulties were related to implementing the core Business Logic**

Let $H_a : G_b \neq G_e$, there was no significant difference ($\rho = 0.6856$) in the scores for the baseline ($\bar{x} = 3.00$, $\sigma = 1.83$) and the experimental ($\bar{x} = 2.50$, $\sigma = 1.29$) conditions, as seen in table 5.6.

DP8 **Concerning this particular setup, most of my difficulties were related to fixing compilation and runtime errors**

Let $H_a : G_b < G_e$, there was a significant difference ($\rho = 0.0188$) in the scores for the baseline ($\bar{x} = 3.25$, $\sigma = 1.71$) and the experimental ($\bar{x} = 5.00$, $\sigma = 0.00$) conditions, as seen in table 5.6. As expected, the participants in baseline group spent more time working with the IDE to create views and the code suggestions (type checking, package imports, among others) before compiling which helped to reduce the compilation errors. The experimental group was working with the console and a text editor.

5.3.6 **Usability Satisfaction**

Once the experiment was completed, the experimental group questionnaire contained additionally a set of questions regarding the usability of agile-scala-android. These were asked after all the
tasks were completed having the participants been faced with problems that they had not foreseen. We now proceed to discuss these questions below.

<table>
<thead>
<tr>
<th>Experimental</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12345</td>
<td></td>
</tr>
<tr>
<td>US1</td>
<td>5.00</td>
<td>0.00</td>
</tr>
<tr>
<td>US2</td>
<td>3.75</td>
<td>1.26</td>
</tr>
<tr>
<td>US3</td>
<td>4.25</td>
<td>0.96</td>
</tr>
<tr>
<td>US4</td>
<td>4.50</td>
<td>0.58</td>
</tr>
<tr>
<td>US5</td>
<td>3.25</td>
<td>1.50</td>
</tr>
<tr>
<td>US6</td>
<td>4.00</td>
<td>0.82</td>
</tr>
<tr>
<td>US7</td>
<td>5.00</td>
<td>0.00</td>
</tr>
<tr>
<td>US8</td>
<td>1.25</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Table 5.7: Succinct overview of the Usability Satisfaction results, containing the averages and standard deviations.

US1 I found the tool commands easy to use.

The experimental group gave a score of $\bar{x} = 5.00$, $\sigma = 0.00$, as seen in table 5.7. The responses gathered were positive and very favourable.

US2 I had no trouble executing and understanding the effects of running the command 'npa'.

The experimental group gave a score of $\bar{x} = 3.75$, $\sigma = 1.26$, as seen in table 5.7. The responses gathered were positive and favourable.

US3 I had no trouble executing and understanding the effects of running the command 'generate'.

The experimental group gave a score of $\bar{x} = 4.25$, $\sigma = 0.96$, as seen in table 5.7. The responses gathered were positive and very favourable.

US4 I had no trouble executing and understanding the effects of running the command 'scaffold'.

The experimental group gave a score of $\bar{x} = 4.50$, $\sigma = 0.58$, as seen in table 5.7. The responses gathered were positive and very favourable.

US5 I had no trouble executing and understanding the effects of running the command 'migrateDatabase'.

The experimental group gave a score of $\bar{x} = 3.25$, $\sigma = 1.50$, as seen in table 5.7. The responses gathered were positive and favourable.
I had no trouble executing and understanding the effects of running the command ‘checkPermissions’.

The experimental group gave a score of $\bar{x} = 4.00$, $\sigma = 0.82$, as seen in table 5.7. The responses gathered were positive and very favourable.

I will most likely use this tool in the next project Android project.

The experimental group gave a score of $\bar{x} = 5.00$, $\sigma = 0.00$, as seen in table 5.7. The responses gathered were positive and very favourable.

If this tool was available for ...

This question had another set of possible answers, listed as follows:

(1) ... free, I would adopt it in future projects;
(2) ... €4.99, I would adopt it in future projects;
(3) ... €9.99, I would adopt it in future projects;
(4) ... €14.99, I would adopt it in future projects;
(5) ... €29.99, I would adopt it in future projects.

The experimental group gave a score of $\bar{x} = 1.25$, $\sigma = 0.50$, as seen in table 5.7. The responses gathered were negative and very unfavourable. Selling this tool is probably not the best approach. Frameworks like Play! and Ruby on Rails are free and open-source. However, there are companies that sell commercial support, and it is the author’s belief this could be the best approach to monetize this tool.

Overall, the feedback regarding the framework was positive and favourable. These results show that agile-scala-android despite being in prototype stage is already very much capable and useful. These results will also be used to improve and iterate on the framework, prioritizing the work.

5.4 Objective Measurement

The empirical analysis of software code quality is not trivial and should be done using common metrics. In 1994, Chidamber and Kemerer proposed six Object-Oriented complexity metrics [CK94]: weighted methods per class, coupling between object classes, response for a class, number of children, depth of inheritance tree and lack of cohesion of methods. Initially, the plan was to use these metrics as an objective measure of validation. However, some tasks were deliberately very hard to complete within the time limit. This meant that it would be incorrect to

5 The database feature should be worked on first, for example.
compare two solutions, when one of them is not even complete. As such, other forms of objective evaluation were considered.

5.4.1 Duration of tasks

The tasks duration is an important metric as one of the goals of the framework is to create a faster development environment. The table 5.8 provides an overview of the time it took for participants to complete the given tasks.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[0, 10] [10, 30]</td>
<td>[0, 10] [10, 30]</td>
</tr>
<tr>
<td></td>
<td>[30, 60] [60, ∞]</td>
<td>[30, 60] [60, ∞]</td>
</tr>
<tr>
<td></td>
<td>τ</td>
<td>σ</td>
</tr>
<tr>
<td>T1</td>
<td>3.50</td>
<td>1.00</td>
</tr>
<tr>
<td>T2</td>
<td>7.00</td>
<td>4.32</td>
</tr>
<tr>
<td>T3</td>
<td>60+</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>56.50</td>
<td>15.89</td>
</tr>
<tr>
<td>T5</td>
<td>20.50</td>
<td>9.40</td>
</tr>
</tbody>
</table>

Table 5.8: Succinct overview of the duration of tasks in minutes.

T1 — Generate a new project

The baseline (τ = 3.50, σ = 1.00) and the experimental (τ = 3.00, σ = 0.82) groups had similar performance on this task, as seen in table 5.8.

This was expected as IDE already solves this problem very well. Nevertheless, this is an significant task since we are also comparing Java versus Scala environments.

T2 — Implement Models

The baseline (τ = 7.00, σ = 4.32) and the experimental (τ = 4.50, σ = 1.29) groups had slightly different performance on this task, as seen in table 5.8.

The experimental group performed slightly faster and was more consistent.

T3 — Implement Views and Controllers

The baseline (τ > 60) and the experimental (τ = 5.75, σ = 2.36) groups had highly different performance on this task, as seen in table 5.8.

As expected, the baseline group had trouble completing this task whereas the experimental group students managed to complete everything under ten minutes, using the scaffold feature present in agile-scala-android.

T4 — Use a database to store Products

The baseline (τ = 56.50, σ = 15.89) and the experimental (τ > 60) groups highly different performance on this task, as seen in table 5.8.
In this task, the experimental group was not capable of executing the task with success. This might be due to the following reasons: (i) incomplete tutorial documentation on database — the tutorial documentation did not mention migrations and how they work —, (ii) poor Slick documentation — the package imports that bring the Implicit conversions for the SQL Drivers are not highlighted in the Slick documentation —, and (iii) lack of imports already included in generated source code. These issues need to be addressed and taken into consideration for the next release of agile-scala-android.

This task might have had yielded different results if the developers in the experimental group were more familiar with Slick and the required database was more complex, with foreign relationships between Notes and Products.

**T5 — Extra functionality**

The baseline ($\bar{x} = 20.50$, $\sigma = 9.40$) and the experimental ($\bar{x} = 26.50$, $\sigma = 10.60$) groups had slightly different performance on this task, as seen in table 5.8.

This unexpected result can be explained on how the subjects in the experimental group solved the problems. Instead of using Androids class AsyncTask\(^7\), they used Scala’s Future\(^8\), which they were unfamiliar with. This different approach resulted in fewer lines of code, but it took longer to develop. Additionally, while the baseline group performed faster in this task, they also experienced more runtime crashes (see section 5.4.4). It is still the author’s belief that the automatic resolution of permissions can save time, and perhaps a more diverse and extensive list of tasks would show its effectiveness.

### 5.4.2 Graphical User Interface completeness

Task 3 requested a GUI to have the CRUD operations over the model Note and Read operation over the model Product. The solution for this problem varied according to which group the subjects belonged to:

(a) if in the baseline group, they used the IDE facilities to develop the interface visually, and then write controller.

(b) if in the experimental group, they used the scaffold feature.

Table 5.9 provides the completeness of the GUI implementation between the two groups.


\(^8\)For more information on Scala’s Future, consult [http://docs.scala-lang.org/overviews/core/futures.html](http://docs.scala-lang.org/overviews/core/futures.html).
An Academic Quasi-experiment

<table>
<thead>
<tr>
<th>Note</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>C R U D</td>
<td>$\bar{x}$</td>
</tr>
<tr>
<td>Baseline</td>
<td>2</td>
</tr>
<tr>
<td>Experimental</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.9: Overview of the Graphical User Interface completeness.

Every element in the experimental group was to complete every necessary views (and controllers), whereas the baseline struggled to produce a usable (even if minimal) interface within the sixty minute time limit.

### 5.4.3 Application startup time

All the applications developed by the students perform very similar in terms of startup time. They all open within one second in a Nexus 7 (2014).

We did not find any noteworthy difference between Scala and Java applications. This correlates with the study that compared Java and Scala performance on Android [DN13].

### 5.4.4 Runtime crashes due to lack of permissions during development

The last task was designed to test the programmer’s ability to know and add the needed permissions. Even when it was clear that a permission was needed (the documentation they found so stated), the participants would sometimes forget to include it, quickly launching the application, only to have a runtime error.

Table 5.10. compares the number of crashes between among the participants in the two groups.

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crashes</td>
<td>Crashes</td>
</tr>
<tr>
<td>$S_a$</td>
<td>2</td>
</tr>
<tr>
<td>$S_b$</td>
<td>5</td>
</tr>
<tr>
<td>$S_c$</td>
<td>5</td>
</tr>
<tr>
<td>$S_d$</td>
<td>2</td>
</tr>
<tr>
<td>$\bar{x}$</td>
<td>3.50</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>1.73</td>
</tr>
</tbody>
</table>

Table 5.10: Overview of the runtime crashes experienced by the participants during development.

The last challenge of the task was to download a file and display its content. The code to accomplish this needs to be on another thread. Indeed, adding permissions automatically did not prevent all runtime crashes, but the experimental group experienced less than half in average.
5.5 Validation Threats

As with any empirical study, there are several validation threats that need to be considered and reflected upon. These can influence the outcome of the results and should be identified and have their impact minimized or mitigated as much as possible. If they are not considered, the results could be invalid in the worst case scenario and ultimately be discarded [Fer10].

A list of validations threats considered and their actual impact for this experiment are reflected as follows:

- **Observation procedure.** The observation procedure used to document the experiment, a screen recording program, could have influenced the participants performance. The results from the question EF1 (section 5.3.3) allow this threat to be eliminated.

- **Lack of motivation.** Considering that the necessary time to complete the experiment was about three hours, participants could start to feel less motivated to complete the tasks. Additionally, there was no reward set for doing this experiment. The results gathered from question EF2 (section 5.3.3) allow this threat to be safely eliminated as participants enjoyed the experiment.

- **Insufficient number of participants.** There were a total of eight participants in this experiment. The value of the study is naturally influenced by the number of participants. In addition, the subjects were also only students. It is the author’s consideration that this was an *in vitro* experiment, and the results are positive indicators that the tool has at least some value. The next experiment could be *in vivo* — that is, a larger scale and with industry professionals doing a larger project with larger timespan. This means that this threat can be discarded and the validity of the results be put in perspective taking into account the number and technical expertise of the participants.

- **Poorly managed expectations.** The experiment was estimated to take about three hours. However, the author believed that the participants would perform the tasks faster, and consequently told the participants of this group the experiment would take only about two hours. This could have influenced the result of question S2 (section 5.3.4). This threat however did not necessarily pose a problem as none of the participants had to stop in the middle of the experiment and so it can discarded with moderate confidence. Nonetheless, future experiments should take into consideration that the expectations need to be more accurate and more realistic. Also, if the time limit is set too high, this might influence the subjects into thinking they completed the tasks very fast.

5.6 Conclusion

This chapter carefully discussed the quasi-experiment conducted in a controlled and synthetic environment to assess the usefulness and effectiveness of the framework developed.
An Academic Quasi-experiment

The pre-test evaluation guaranteed that there was no statistical deviation among the two groups, concerning their background and basic skills with respect to the programming language used. The post-test questionnaires gathered additional information and were used to evaluate the outcomes of each treatment. An objective evaluation was conducted on several metrics (the time, the completeness of the GUI and the number of runtime crashes due to lack of permissions). The final results support the hypothesis that using agile-scala-android provides an overall faster development environment. Not all of the initial hypothesis held though, as some areas that needed additional work were identified, mainly the database feature and its documentation in the tutorial.

In general, the execution of this quasi-experiment will allow the agile-scala-android tool to mature and a new experiment will have to be conducted with different participants to assess if it has evolved. This new experiment should incorporate the knowledge from the identified validation threats.
An Academic Quasi-experiment
Chapter 6

Conclusions and future work

This chapter presents an overview of the project, the main contributions and possible future work.

6.1 Overview

This is a dissertation report that described the project’s context, motivation and goals, and puts it in perspective with the current state of the art. It exposes the research and the building of agile-scala-android. It documents the experiment performed and discusses the results.

6.2 Contributions

The following items summarize the main results obtained:

1. **Web Patterns research for rapid prototyping.** A few distinctive patterns were analysed in the context of rapid prototyping and with their value if applied to Android development.

2. **A complete framework to support Android development with Scala.** The most important contribution of this dissertation was agile-scala-android — an usable framework that provides a complete Android development environment.

3. **A tutorial using Typesafe’s Activator.** No matter how good a tool is, without proper documentation no one will be able to use it. This tutorial was an unconventional approach that had mostly positive results and feedback.

4. **An Academic Quasi-Experiment.** It was important to understand if the tool was indeed usable and if the features it possesses are valuable for, and have impact in the development workflow. The outcome of this experiment was valuable, not only as it corroborated most hypothesis but as it laid ground for prioritization of future work.
Conclusions and future work

5. **Result dissemination.** Asides from this document, result dissemination took form in two paper submissions for PLoP and INForum. As of time of writing, these papers are still in the submission process, under revision.

### 6.3 Future work

The implementation of the framework is currently in a prototype stage. There is considerable amount of work to evolve the current plugin to a final and complete solution. The next steps in terms of development and features considered relevant are the following:

1. **Implicit Scaffold.** Currently, scaffolding is performed when the programmer so requests it. It would be interesting to explore the possibility of automatically having all the models scaffolded whenever the project is compiled. This would mean the programmer could change the models freely, and use these views and controllers if required.

2. **Futher IDE Integration.** IDEs are a very good environment for any kind of software development. At some point, these features could be part of an IDE. Alternatively, the tool could evolve to become a plugin for a specific IDE.

3. **Real-world testing and mature current implementation.** The usefulness of the features present in agile-scala-android is dictated by real world usage. Having this in mind, the future work should reflect this and be prioritized by the users expectations and needs.

4. **Swagger\(^1\) Restful API Integration.** Android applications are typically built as another source of information for a particular website or webservice. As such, using Swagger, an API could be imported automatically: need models generated and a API class file that would hold the functions needed to interact with this service.

### 6.4 Final Remarks

In an Android dominated world, there is considerable amount of work to be done to promote the use of Scala programming language.

Patterns in Web development were looked at, the state of the art for existing tools and projects, dissected Web Frameworks for possible similarities with Android and built a complete framework. An experiment was conducted and found that these patterns, if applied to Android, can indeed have a positive impact throughout the application development life cycle.

From our experience, it is the authors’ belief that agile-scala-android can serve as a basis or as inspiration for more sophisticated work to further improve the development processes of not only Android, but other interactive applications.

\(^1\)Swagger is an open standard for describing Restful APIs. For more information consult: [https://hulloverb.com/developers/swagger](https://hulloverb.com/developers/swagger).
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


Appendix A

Typesafe’s Activator Tutorial

The following images are screenshots taken directly from a project opened inside Typesafe’s activator with the tutorial for agile-scala-android.
Typesafe’s Activator Tutorial

Figure A.1: Screenshot of the *Getting Started* tutorial section.
Typesafe’s Activator Tutorial

Figure A.2: Screenshot of the New (Scala) Project Android tutorial section.
Typesafe’s Activator Tutorial

Figure A.3: Screenshot of the Creating a new Model tutorial section.

The next logical step after creating a new application is defining the business models that it will work with. For this purpose, there's an auxiliary command that manages this, in a very similar way that Ruby on Rails. The command is the following:

```
generate <Model Name> <attributes>
```

The attributes are a tuple of name with type. For example, the following commands creates a new model called Post, with title, number of views, and date as their attributes.

```
generate Post title: String numberOfViews: Int date: Date
```

Please use camel-casing when naming the variables as it will be important for other commands.

At this point, I want you to open a console at the project root, start SBT, and execute this command yourself. After this, the following file was generated:

```
src/main/scala/pimenteiroseca/activatorexample/models/Post.scala
```

This is a simple Scala case class, and you can edit it and expand it further.

Every generated model will have a database mapping that is automatically generated. This feature will be explained in the next section.
Typesafe’s Activator Tutorial

Figure A.4: Screenshot of the Database tutorial section.
A basic Android application has no permissions associated with it by default, meaning it cannot do anything that would adversely impact the user experience or any data on the device. This means to access common resources such as GPS, Camera, Internet, a permission needs to be added to the manifest.

Usually, only at runtime will an error message inform the developer about the missing permission. To solve this problem, the Agile Scala Android plugin analyzes the source code and adds known missing permissions automatically.

Let’s test this feature now:

1. Open the MainActivity.
2. Add to onCreate function the following code:
   ```scala
   import android.content.Context
   import android.net.wifi.WifiManager
   val wifi = getSystemService(Context.WIFI_SERVICE).getInstance(WifiManager)
   wifi!!.disconnect()
   ```
3. Go ahead and run the project
4. Open the manifest and confirm that the permissions were added.

Internally, this mostly achieved by looking at the application’s call graph and comparing to the results of Proguard, which contains a mapping between permissions and specific Android’s functions.

It should be noted that this technique might miss a permission or add some unnecessarily, before deploying to production the application, the permissions list should be reviewed.

Figure A.5: Screenshot of the **Automatic Android Permission resolution** tutorial section.
Figure A.6: Screenshot of the *Scaffolding* tutorial section.
Figure A.7: Screenshot of the *Wrap up* tutorial section.