PROJECTE FINAL DE CARRERA

ANKIDROID: OPEN SOURCE DEVELOPMENT IN ANDROID

ESTUDIANT: Eduard Zamora Suriñach
DIRECTOR: Alejandro Rios Oruña
PONENT: José Ramón Herrero

DATA: 20 de Gener de 2011
Dedicated to the memory of my grandfather, Josep Suriñach Font, who, with his example, has taught me what being successful in life is really about.
First of all, I would like to thank all the people involved in the AnkiDroid project or that participated at some point in it, developers, contributors, translators and users alike. Specially, to Damien Elmes for creating Anki in the first place and, without him being aware, sparking my interest in the topic of efficient memorization and SRS, to Kostas Spyropoulos for bringing new life to the community and for proofreading this thesis and to Nicolas Raoul, for convincing me that AnkiDroid would make a great project for a thesis. I would need pages and pages to list all the people that participated in some way with the project, and although I cannot thank you one by one here, you know who you are and you should know that I really appreciate your participation.

Second, I would like to thank all my coworkers and at the same time friends at Mobivery, for staying with me during the whole development of this project. In particular, many thanks to my director Álex Ríos and to Alejandro Scandroli for letting me choose a project of my particular interest and for trusting in me, to Javier Camarasa for his enthusiasm and honest excitement about the topic of this thesis, to Albert Hernández for his help with the structure of the thesis, to Isabel Blas and Helena Poch for their invaluable advice on design aspects and to Víctor Fernández for constantly nagging me to finish writing this thesis. Although stressful at times, I know that his intentions were good: he really could feel my pain and he only wanted the best for me.

Also, I would like to show my immense gratitude to my friends, Daniel López, Jordi Chacón, Lorena Manzano, Iván Navarro, José Antonio Camallonga, Sergi Boada, Veronica Mogort, Rubén Enrich and Lorena Pérez for holding me accountable, for always encouraging me, for helping me disconnect and take a break once in a while and for neither having a bad word or complain in all this time, in spite of the few time I had for them.

Finally, I would like to say thank you to all my family, that supported me through all these years. They have been the constant and the safety net that has allowed me be where I am.

This would never have been possible without you.

Eduard Zamora Suriñach
# Contents

1 Introduction 8
   1.1 Where the idea for this project came from? 8
   1.2 Project description 9
   1.3 Project goals 10
   1.4 Thesis organization 10

2 Agile Development with Scrum 13
   2.1 What is Scrum? 13
   2.2 Why Scrum was chosen? 15
   2.3 How we do Scrum 16
      2.3.1 Roles 16
         2.3.1.1 Product Owner 16
         2.3.1.2 Scrum Master 17
         2.3.1.3 Team 17
         2.3.1.4 Stakeholders 18
         2.3.1.5 Managers, executives, commercials 18
      2.3.2 Tools 18
         2.3.2.1 Triple Constraint Management 18
         2.3.2.2 Timeboxing 20
         2.3.2.3 Taskboard 20
         2.3.2.4 Scrum Cards 22
         2.3.2.5 Post-its 23
         2.3.2.6 Google Documents 23
      2.3.3 Artifacts 24
         2.3.3.1 Product Backlog 24
         2.3.3.2 Sprint Backlog 24
         2.3.3.3 Burndown Chart 24
         2.3.3.4 Impediment Backlog 25
         2.3.3.5 Kaizen Backlog 25
      2.3.4 Meetings 25

3
2.3.4.1 Product Planning ........................................... 26
2.3.4.2 Sprint Planning ........................................... 26
2.3.4.3 Daily Scrum ............................................... 27
2.3.4.4 Demo ....................................................... 27
2.3.4.5 Retrospective ............................................. 27
2.3.5 How we estimate ............................................. 28
  2.3.5.1 User stories ............................................ 28
  2.3.5.2 Story points ........................................... 28
  2.3.5.3 Planning Poker ........................................ 29

3 Android ......................................................... 30
  3.1 What is Android? ........................................... 30
  3.2 Application Components .................................. 31
    3.2.1 Activity ............................................... 31
    3.2.2 Service ............................................... 33
    3.2.3 Broadcast Receiver .................................... 36
    3.2.4 Content Provider ...................................... 37

4 Anki ............................................................. 38
  4.1 What is Anki? ............................................... 38
  4.2 Cognitive Psychology Concepts Used by Anki ............ 39
    4.2.1 Active Recall ......................................... 40
    4.2.2 Spaced Repetition .................................... 40
  4.3 Basic Terminology and Usage .............................. 41
    4.3.1 Cards .................................................. 41
    4.3.2 Decks .................................................. 44
    4.3.3 Facts and Fields ...................................... 44
    4.3.4 Card Models or Templates ............................ 47
    4.3.5 Models ................................................ 48

5 AnkiDroid Analysis and Design .............................. 50
  5.1 User Stories ............................................... 50
  5.2 Architecture .............................................. 52
  5.3 DB Diagram ................................................ 54
  5.4 Navigational Maps ........................................ 56
    5.4.1 Open deck ............................................ 58
    5.4.2 Review deck .......................................... 59
    5.4.3 Configure preferences ................................. 60
    5.4.4 Download deck ....................................... 61
    5.4.5 Synchronize deck ...................................... 62
    5.4.6 Synchronize all decks ................................. 63
6 AnkiDroid Implementation

6.1 Development Environment
   6.1.0.1 Android SDK
   6.1.0.2 Git
   6.1.0.3 GitHub
   6.1.0.4 Eclipse
   6.1.0.5 ADT plugin

6.2 The Android Manifest

6.3 User Interface

6.4 SRS
   6.4.1 Getting the next card to review
   6.4.2 Answering a card

6.5 Download Manager

6.6 Synchronization

6.7 i18n

7 AnkiDroid Open Source Community

7.1 What Does an Open Source Project Need?
   7.1.1 People
   7.1.2 Website
   7.1.3 License
   7.1.4 Features and Requirements List
   7.1.5 Version Control
   7.1.6 Issue Tracker
   7.1.7 Communication channels
   7.1.8 Documentation
   7.1.9 Translation strategy

7.2 AnkiDroid Community Infrastructure

7.3 Licensing
   7.3.1 Choosing a License
     7.3.1.1 Options
     7.3.1.2 Principal Open Source Licenses
   7.3.2 Applying a GPL v3.0 License

8 Planning and Economic Analysis

8.1 Timeline

8.2 Economic Analysis


B AnkiDroid Code Conventions

B.1 Introduction ................................. 107
B.1.1 Layout of the recommendations .............. 107
B.1.2 Recommendations importance ................. 107
B.1.3 Automatic style checking and formatting ....... 107
B.2 General recommendations ....................... 108
B.3 Naming conventions ............................. 108
B.3.1 General naming conventions .................. 108
B.3.2 Specific Naming Conventions ................. 111
B.4 Layout techniques ............................. 112
B.4.1 Length line ................................ 112
B.4.2 Indentation ................................ 112
B.4.3 Braces ..................................... 113
B.4.4 White spaces ................................. 113
B.4.5 Blank lines ................................ 114
B.5 Control structures ............................. 114
B.5.1 if ........................................... 114
B.5.2 for .......................................... 114
B.5.3 while ....................................... 115
B.5.4 do-while .................................... 115
B.5.5 switch ....................................... 115
B.5.6 try-catch ................................... 116
B.6 Comments ...................................... 116
B.7 Logging ........................................ 118
B.8 File organization ............................... 119
B.9 Miscellaneous ................................ 121
B.9.1 General ...................................... 121
B.9.2 Types ........................................ 122
B.9.3 Variables and constants ...................... 122
B.9.4 Operators .................................... 123
B.9.5 Conditionals ................................ 123
B.9.6 Loops ........................................ 124
B.9.7 Methods ..................................... 124
Chapter 1

Introduction

1.1 Where the idea for this project came from?

Back in August 2008, I was in the KTH library with one of my best friends from FIB (Facultat d’Informàtica de Barcelona), where we were drilling over and over again all irregular Swedish verbs, in an attempt to stick them into our heads. For those wondering, KTH stands for Kungliga Tekniska Högskolan (Royal Institute of Technology, in English) and is one of the leading technical universities in Europe, situated in Stockholm, Sweden, where we were doing our Erasmus. Our Swedish examination was only one week away... and we still had a lot of studying and memorizing to do if we wanted to pass.

Back to our studying, I was getting increasingly irritated with the inefficiency of rote memorization, and as a consequence a thought popped into my head: “There has to be a better way to memorize than this.” So, I took my laptop and started to look for that way. I don’t remember where that search started, but I do remember where it finished: in the Wikipedia page of a software called Anki, which incorporated an smart algorithm that supposedly made memorization easier.

It was not until after our Swedish examination (that just for the record, both of us passed with good grades), that I started to use Anki correctly for what was my main interest at the time: learning English. A month later, after reviewing correctly in my first try a word that had seemed so strange when I first saw it, I decided to check my progress using Anki: more than 500 words learned in less than 3 hours in total time convinced me that I was into something there.

The only thing that bothered me about Anki is that I could only take advantage of it when I was at home, in front of the computer, and therefore I had not the chance to transform these wasted moments during the day in useful ones.

Almost a year later, once I finished all the courses of my degree I started to look for a company where I could do my final project (PFC, or Projecte de Final de Carrera). I was highly interested in developing mobile applications, I think for the whole new arrange of possibilities that running applications on the go allows, and I found the best place to do it in Spain: Mobivery. When they gave me the freedom
to choose my own project, I connected the dots and I knew without a doubt what I
wanted to do: I would make an application to study efficiently no matter where you
were.

1.2 Project description

The project at hand revolves around a variety of main topics: agile development,
open-source, Android and SRS software (SRS stands for Spaced Repetition Soft-
ware, and it is based in the spaced effect phenomenon discovered in cognitive
psychology).

Combining all these topics, the project will describe and document the work that
has been done defining and implanting a development process for Android appli-
cations, from taking requirements to releasing the final application to the Android
Market. All of this, using Scrum like agile methodology for personal organization
and working side by side in an open-source environment, that has been created
from the ground.

The project will also explain briefly then, the characteristics and philosophies
behind agile development in general, and Scrum in particular.

The point where all the topics mentioned before converge is in the practical ex-
ample that was used for the project: AnkiDroid. AnkiDroid is the Android version
of Anki, an SRS software multi-platform (Mac OS, Linux, Windows...), open-source
and free that, at the time of beginning the project, still did not have an Android ver-
sion. In brief, the main goal of both Anki and AnkiDroid is to make the study for long
term memorization of any subject as efficient as possible, minimizing the number
of times the information has to be seen and the time spent reviewing, while maxi-
mizing the retention of the material. All of this is achieved with an implementation
of an SRS algorithm.

A very interesting thing about AnkiDroid is that it touches a lot of different pro-
gramming topics: it makes an intensive use of databases (SQLite in this case),
both for preparing the study sessions and for updating the statistics and recalculat-
ing when the information has to be shown again, it has work on testing (both
functional and unitary), UI design, usability and implementation, it handles images
and audio (and in the future, also video), it is internationalized to 20 languages by
now, and it uses connections in order to download new material for study (a Down-
load Manager has been implemented for that) or synchronize your material and
your progress with the rest of Anki clients.

Furthermore, AnkiDroid also involves work creating, organizing and putting in
place all the tools, elements and documents needed for the correct functioning of
the open-source community that has been created around it, and taking care of the
relationships formed with our users.

All of this made it, all in all, a very complete and rewarding project to work for.
1.3 Project goals

From the project description made in the prior section and having in mind that we have to set a scope in order to delimit the work that will be done, the next goals are derived:

- Investigate, study and document the Android platform and its more important components and characteristics.
- Study and document the main characteristics of agile development.
- Study and investigate about the open-source movement.
- With the information learned about open-source, set a solid foundation for the open-source community working on AnkiDroid, creating the needed tools, organizing the workflow, setting and defining the communication streams and writing the necessary documentation.
- Development of AnkiDroid, following Scrum in order to organize my personal work and at the same time organizing myself and collaborating with the rest of the AnkiDroid community. Damien Elmes, the developer of Anki, spent more or less four years in that enterprise (and still going on) so it would be unrealistic for my part to try to completely finish AnkiDroid in the time available for a final project. Due to this limitation, the scope is limited to make AnkiDroid completely suitable for reviewing (and not for creating material). Anyway, this scope could vary greatly depending on the amount of participation that the project attracts.
- Development of an open and close relationship with our users.

1.4 Thesis organization

The organization of this thesis, and what can be found in each chapter, is as follows:

Chapter 1: Introduction

The chapter at hand serves as an entry point for the thesis, giving an overview to the reader about how the idea was generated, what this project is about and why it was chosen, the goals we set for it and how the thesis is organized and structured.

Chapter 2: Agile Development with Scrum

The second chapter of the thesis provides a walk through the methodology used during the whole project, Scrum. A justification about why Scrum was selected for this project along with its characteristics, roles, tools needed and an example implementation can be found here. If you are interested in agile development in
general or Scrum in particular, this chapter should provide the needed information to grasp their philosophy and start applying it.

Chapter 3: Android

The third chapter describes briefly what Android is and the most important components used in Android development. It intends to give the reader a general idea about how the platform works and hopefully provide him with the necessary knowledge to understand the rest of the thesis, specially the chapter about implementation. By no means it aspires to teach Android development, which would require an entire book. If you are interested in that, you can look for the Android development books used in the bibliography.

Chapter 4: Anki

This chapter is dedicated to Anki, the software that inspired all this project. In it we explain what is Anki, how it works and the basic vocabulary it uses, that will be useful to understand completely the rest of the project. Also, an explanation about what SRS is and the cognitive psychology research it is based on is provided.

Chapter 5: AnkiDroid Analysis and Design

Displays the most important aspects included in the phases of Analysis and Design of the iterative and incremental process.

Chapter 6: AnkiDroid Implementation

Chapter that discusses the implementation of the most important features found in AnkiDroid, trying to not overwhelm the reader with excruciating detail. In case, someone wants to know more about the actual implementation, the source code can be consulted.

Chapter 7: AnkiDroid Open Source Community

Chapter 7 points what the main goals of the majority of open source projects are, and what infrastructure, tools and documentation usually they need in order to run successfully. After that, we explain how the infrastructure of the AnkiDroid community looks like, justifying the decisions made at every point. Furthermore, an explanation about how to choose and apply an open source license is included.

Chapter 8: Planning and Economic Analysis

This chapter contains the final planning, the distribution of my work and an estimation of the production cost of the project.
Chapter 9: Conclusions

Includes the personal conclusions and learnings I extracted undertaking this project. Additionally, some pointers for future work in order to improve both the application and the functioning of the community as a whole are given.

Appendix A: AnkiDroid Manifest

The Manifest file of the AnkiDroid application, one of the most important files in every Android application, attached as a reference and example.

Appendix B: AnkiDroid Code Conventions

Code guidelines defined and selected from various distinguished sources, adapted to the necessities of the project and approved by the initial developers. It is one of the most important documents for the open source community, and the one that has taken more time to create.
Chapter 2

Agile Development with Scrum

The main goal of this chapter is to show the reader the basics of Scrum: what exactly is, in what contexts does using Scrum make sense and how it can be implemented. This could be done in a lot of ways, but like there is already a large amount of good material explaining the theory behind Scrum, and inspired by the great book *Scrum and XP from the Trenches* [1], the main focus of the chapter is on the practical side of it.

At the beginning of the chapter then, a general and short overview of what Scrum is will be presented to the reader, showing its main principles and characteristics.

After that, we will discuss the reasons why we considered Scrum the development methodology that best suits the project at hand, and therefore why it was the selected one in front of other methodologies.

Finally, we will show how we, at Mobivery, implement our own version of Scrum.

Also, doing all that, I hope to provide the reader with enough knowledge and interest in Scrum to consider using it in his future projects, whenever its characteristics and circumstances make it a good choice.

2.1 What is Scrum?

Scrum is an agile development methodology created from the collaboration of Ken Schwaber, Jeff Sutherland, John Scumioitalas and Jeff McKenna in the early 1990s. Its name was taken from an article written by Takeuchi and Nonaka, who compared the high-performing, cross-functional teams to the scrum formation used in Rugby, an analogy meaning that, although the individuals had different responsibilities and skills, the team as a whole has to work together, push together, in order to achieve the common goal.

Being an agile methodology, we can see how Scrum practices and guidelines were formed and fed from its source, the *Agile Manifesto* and its principles. In
case the reader is not familiar with them, and because we consider that they are interesting and important to completely grasp the spirit and motivation of Scrum, we cite them here:

**Agile Manifesto** [2]

*We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

*Individuals and interactions over processes and tools*  
*Working software over comprehensive documentation*  
*Customer collaboration over contract negotiation*  
*Responding to change over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

And derived from the manifesto above:

**Agile Principles** [3]

1. *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*

2. *Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.*

3. *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*

4. *Business people and developers must work together daily throughout the project.*

5. *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*

6. *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*

7. *Working software is the primary measure of progress.*

8. *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. Continuous attention to technical excellence and good design enhances agility.*

9. *Simplicity—the art of maximizing the amount of work not done—is essential.*

10. *The best architectures, requirements, and designs emerge from self-organizing teams.*

11. *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*
These principles can be seen (among many other aspects) in the fact that Scrum relies on cross-functional and self-organizing teams in order to get the work done and that it fully accepts that there is always uncertainty, changes are going to happen and we cannot know everything we need in advance. That also makes Scrum an ideal methodology in environments where exists a lot of uncertainty and where the requirements are unstable.

Also, like many other agile methodologies, Scrum relies heavily on the concept of iterative and incremental development, that means that, contrary to methodologies that follow the waterfall model, the system or product is developed through a series of repeated cycles (iterations) and working on small portions in each cycle (increments).

An iteration in Scrum is called sprint, usually lasts between one and four weeks, and its main goal is to create a new potentially shippable version of the product.

Later we will explain the process that the Scrum Team follows in each sprint, the meetings that it holds and the artifacts that it generates.

For now, we only need to know that at the beginning of the sprint, the team decides which features they will work on and then the sprint becomes fixed and frozen. Nothing else can enter the sprint and disturb the focus of the team. Once the sprint finishes, the team shows the work completed and using the experience of that sprint and reflecting about what happened, try to come up with ways to improve in the future.

\[\text{2.2 Why Scrum was chosen?}\]

At the time of choosing a development methodology for a project its characteristics and circumstances have to be considered, in order to find a methodology that takes them into account and adapts to them. This is an important decision because the election of a suitable methodology can decide the outcome of the project, either for good or for bad.

In our case, was decided that Scrum was the best fit for AnkiDroid for the next reasons:

\begin{itemize}
  \item Complete ignorance of the Android platform prior starting the project.
  \item Difficulty assessing how much work the functionalities planned would really take and all its implications.
  \item Very little knowledge about open source.
  \item High uncertainty about how many people were going to collaborate with the project.
  \item Uncertainty about which tasks would the rest of the people collaborating with the project undertake, making long term planning near impossible.
  \item Familiarity with Scrum, which was already established at work.
\end{itemize}
As we can see, one of the defining characteristics about AnkiDroid, in terms of planning and organizing its work, is that it has a lot of uncertainty. In a case like that, a rigid or predictive methodology makes no sense at all because all the planning done could prove useless in less than a month, for a variety of reasons: being the project open source it encourages the participation of volunteers in any kind of task, but volunteers come and go on unpredictable basis and work mainly in whatever they feel like to. These collaborations are impossible to foresee and plan and could conflict with any long term planning done.

That leaves us with agile methodologies like the best candidates, due to its adaptability to changes both in priorities and requirements, and the familiarity we already had with Scrum gave it the edge to be the selected one.

2.3 How we do Scrum

Now that we have a general idea about what Scrum is, this section intends to show how it is practiced, using our work at Mobivery as a real world example, and learning anything we need along the way.

On the other hand, the reader has to take into account that there is not a unique and correct way to do Scrum, and this is no exception. We only present here a real world example because, in our opinion, a lot more can be learned from a war story than from theory alone, and with the hope that the reader could learn both from our successes and from our many mistakes.

2.3.1 Roles

Scrum contains a set of predefined roles: Product Owner, Scrum Master, Team and the rest of the people related to the project somehow. Basically, these roles are divided between the ones that are inside the Scrum Team (Product Owner, Scrum Master and Team) and the roles that are outside the Scrum Team (stakeholders, managers, executives and commercials).

The Scrum Teams are usually small, formed by 5-9 persons (including Product Owner and Scrum Master), and that gives them flexibility and ability to adapt to changes quickly. If a Scrum Team grows too much, there is an overhead in the communications in order to keep everyone in the loop, and the Scrum Team loses part of its efficiency and flexibility. For this reason, in cases like that it is recommended to split the Scrum Team in two, being again in the range of 5-9 persons per team.

2.3.1.1 Product Owner

The Product Owner represents the voice of the customer inside the team. His job is to know what the customer really wants, translate that to customer-centric items (usually known as user stories), prioritize them in order of importance and put them
into the Product Backlog. He has to assure that the team works on the right things, the things that are important to the customer, and in the right order.

Also, another vital function of the Product Owner is that he acts as a filter, protecting the team from external disturbances in the middle of a sprint. All the requests related to changing features or priorities or adding new user stories in a sprint have to be directed to the Product Owner, preserving in this way the focus of the rest of the team. That does not mean that another member of the team can not talk to a customer directly in some circumstances. For instance, if someone in the team needs to clarify a technical issue in order to finish the task that he is doing (which does not modify the work planned for that sprint), it is a better option that he communicates directly with the customer than using the Product Owner as a proxy for the communication.

2.3.1.2 Scrum Master

The Scrum Master is the person inside the team responsible for the correct implementation of Scrum, enforcing the needed rules and assuring that the meetings are held on time.

Moreover, like the Product Owner was a filter for external disturbances, the Scrum Master has to protect the team from internal impediments and keep them as focused as possible in the tasks planned for that sprint. If some member of the team has a problem or impediment that is affecting his productivity (and it is not a matter of the customer trying to make the team work on new features that were not planned for that sprint) he should go talk to the Scrum Master first. If the Scrum Master can not solve his problem, the Scrum Master will direct the concern to the Product Owner or to the managers, as is needed.

It is important to note that the Scrum Master is not the leader or boss of the team, as Scrum Teams are self-organized. He is just another member of the team with additional responsibilities related to the Scrum process.

2.3.1.3 Team

The team is formed by the people responsible for actually implementing and delivering the product or system. So, by that definition, and although she is part of the Scrum Team, we do not consider the Product Owner to be part of the team.

The team has to possess cross-functional skills in order to be able to perform all the work related to a project, all by themselves: design, develop (client, server and in diverse platforms), test, write technical documentation, etc.

Our team is formed by designers, developers and the Scrum Master (who is a developer himself). There is a lot of discussion about whether the designers should be part of the team or they should be treated as an external department, due to the difficulty to integrate their creative work in the Scrum process. We considered that the benefits we would obtain if we could successfully integrate the designers into the Scrum Team far outweighed the inconveniences, so we gave it a shot. And
after some months of adjustments, we can say that the result is satisfactory: having designers on board improves our estimations (because every part of each task is represented and understood) and avoids misunderstandings.

2.3.1.4 Stakeholders

Stakeholders are all the people who the project is made for, and who justify its production.

Stakeholders should communicate closely with the Product Owner in order to agree what the project will do and how. In the middle of a sprint, the major part of their communication have to be with the Product Owner, if the team needs to clarify something in order to complete a task the same team will initiate the conversation with the stakeholders, not the other way around.

They should be directly involved in the Sprint Review or Demo, where they could see the work completed in each sprint and give feedback to the team about it. Unfortunately, we did not completely succeed on this aspect and the participation of the stakeholders in these meetings should be higher, avoiding future misunderstandings thanks to their early feedback and helping the team be more accountable for their work.

2.3.1.5 Managers, executives, commercials

That includes the rest of the people in the organization that set the environment and sign the projects that one of the Scrum Teams will build. As stakeholders, if they want something from a Scrum Team they have to direct their requests to its Product Owner, preserving in this way the team’s focus.

2.3.2 Tools

In this subsection we’ll see the tools, both physical and mental, that we use when practicing Scrum. It is important to say that Scrum does not necessarily require any fancy or expensive tool and everybody, understanding the reason behind each tool and using a little imagination, can find alternatives that fit his concrete needs, preferences and budget.

2.3.2.1 Triple Constraint Management

All projects have to be developed and delivered under certain conditions or constraints. Usually, the constraints that interest most both the company and the customer are:

- Time (or when will the project be finished?)
- Cost (or how much money is this going to cost?)
• Scope (or exactly what work will be done?)

The *Triple Constraint Management* graph displayed below shows the tight relationship between these three constraints:

![Triple Constraint Management Diagram]

Figure 2.1: Triple constraint management.

What this graph is trying to tell us is that one of the constraints cannot be changed without affecting at least one of the other two, there has to be a trade-off between the three of them.

That is to say, the product can not be developed in less time for the same cost and with all the functionalities planned. If the customer really wants to have the project sooner, he has to sacrifice something else: either resign to some feature or pay more in order to put more people working on the project. Or if the customer wants to add more features, the team will need more time to finish and the project’s cost will be higher.

In the real world, the *Triple Constraint Management* is an excellent tool for negotiating both with customers and with the Product Owner: you can let them fix two constraints but the third one has to be decided by the team.

If we observe the graph carefully, we’ll see that in fact there is another constraint: quality. Quality is placed inside the triangle because depends on the relationship between the other three constraints. If this relationship is harmonious, if the tradeoff has been made correctly, we can assure a good level of quality but if we want to change one of the constraints without paying the correspondent tradeoff, quality suffers. And when quality suffers we generate what is known as technical debt, a debt that will have to be paid in the future. For this reason, although the possibility to play with quality as a constraint exists, we don’t recommend it as an option.
2.3.2.2 Timeboxing

Timeboxing is a simple but powerful technique that consists on setting a firm deadline or time limit to perform a task (referring to the *Triple Constraint Management*, timeboxing would be the case where time is fixed).

Its power resides in the fact that knowing that the time available to finish something is finite forces people to be efficient in its use. In a sense, it tries to reverse what is known as *Parkinson’s Law*:

Work expands so as to fill the time available for its completion.

Then, making the time available finite we limit how much work can be expanded or even if it has to be contracted, leaving only its critical parts.

It is applied widely in agile methodologies and therefore also in Scrum, where it is used, for instance, to limit the duration of every meeting. Also, even a sprint is a kind of timeboxing.

2.3.2.3 Taskboard

The taskboard is the place that is used to clearly and visually represent the state of the tasks or user stories planned for the current sprint. That can be done in many, many ways and later we will explain how we do it, but the important thing to keep in mind is that the taskboard has to:

1. Allow anybody, with a quick glance, to know what and how the team is doing.
2. Focus and recenter any member of the team, every time he takes a look at the taskboard.
Figure 2.2: Taskboard used in one of the Scrum teams at Mobivery.

As we can see above, our taskboard is divided in four main columns, representing the state of the tasks in any given moment. These columns are:

- **Pending**: Tasks that have not been started yet.
- **On Progress**: Tasks the team is currently working on.
- **W4F (Waiting for Feedback)**: Tasks awaiting for some information or impediment to be resolved before resuming the work on them again.
- **Done**: Tasks completely finished (is important that the definition of what is exactly done is decided beforehand).

We also use another set of special states, that are indicated by turning upside down a task or user story (that could be perfectly done by adding more columns to
represent these states, but we did not want to overcrowd the taskboard). Depending on which column this is done to, it has a different meaning.

Turning upside down a task or user story on:

- **Pending**: Nobody is currently working on the task, but someone started it before.
- **W4F (Waiting for Feedback)**: Steps to resolve the impediment have already been taken.
- **Done**: It is completed but awaiting for review.

The taskboard is usually updated every day, in the Daily Scrum.

### 2.3.2.4 Scrum Cards

In Scrum, we use cards similar to the ones of Poker in order to estimate the abstract cost of a user story or task. For this reason, this estimation's process using Scrum Cards is also known as *Planning Poker*.

A set of Scrum Cards contains the next cards: 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100, ?, ∞ and coffee.

The cards from 0 to 100 indicate the abstract cost of a task or user story. The card with symbol ? means that the team needs more information in order to be able to give an estimation or that they have no idea about its cost. The card with the ∞ symbol means that it can not be done. Finally, the coffee card means that the person who showed it thinks that everyone would benefit from a short break.
The main advantage of using Scrum Cards when estimating is that it forces everyone in the team to give his honest estimation without the influence of other members of the team.

There are various alternatives to using physical Scrum Cards, from applications for iPhone or Android to using the fingers to represent the desired cost. In any case, we recommend to use one option that allows every member of the team to give his estimation before seeing the one of the others.

2.3.2.5 Post-its

Post-its are simple but very useful in Scrum. They are mainly used to write user stories or tasks on them and their main advantage in front of other alternatives are that they:

- Provide visibility.
- Make changing priorities or updating the state of tasks really easy.

2.3.2.6 Google Documents

Another tool that we use frequently, for its capacity for sharing documents between all members of the Scrum Team, is Google Documents. Google Documents is
useful in order to maintain different metrics about the team’s velocity, focus and efficiency and to store digital copies of the product backlog and the sprint backlog.

In spite of this, it is not as useful as the taskboard for keeping the team on the right track because it lacks the visual aspect of the taskboard and the team has to consciously remember to take a look at it.

2.3.3 Artifacts

Next, we’ll see the artifacts or documents that are generated during the Scrum process, along with a short explanation for each one.

2.3.3.1 Product Backlog

The *Product Backlog* is a prioritized list of all product features (in the form of user stories), bugs and tasks that eventually have to be done by the team. It is managed, maintained and prioritized by the Product Owner.

In spite of this, everybody can add items to the *Product Backlog* at anytime, as long as the Product Owner is aware of it.

2.3.3.2 Sprint Backlog

The *Sprint Backlog* is a prioritized list of all user stories, bugs and tasks that the team has committed to finish in a given sprint. Or in other words, the *Sprint Backlog* is a subset of the *Product Backlog*, maintaining the priority set by the Product Owner.

This one is maintained and managed by the team.

2.3.3.3 Burndown Chart

A *Burndown Chart* is a graph that represents how much work (in our case, calculated in abstract points) there is remaining in a given sprint. It is updated daily at the Daily Scrum and maintained by the team.
2.3.3.4 Impediment Backlog

The Impediment Backlog is a list of actions to do in order to resolve impediments that are preventing the team to perform optimally. Usually each one of these actions is assigned to a component of the team, in order to increase commitment and assure that they will be done.

2.3.3.5 Kaizen Backlog

The Kaizen Backlog is a list of improvements to implement. As with the Impediment Backlog, the concrete improvements are also assigned to a specific member of the team.

2.3.4 Meetings

In this subsection, we will walk the reader through the different Scrum meetings that occur in a sprint, showing their characteristics and how we run them at Mobivery. The meetings are presented following their natural order (that is, Product Planning, Sprint Planning, Daily Scrum, Demo and Retrospective), so the reader can imagine how it would be to work using Scrum for a sprint.
2.3.4.1 Product Planning

The sprint starts officially with the Product Planning, once every two weeks, at 10am on Monday. In this meeting, the Product Owner presents and explains the new projects to the team. Then, the complete Scrum Team identifies and defines the user stories for each project, generating in that way the Product Backlog. At the same time that the user stories are being defined, the team gives a first estimation for each one of them, using Planning Poker, and the Product Owner sets their priority. It has a timeboxing of two hours and a half and it is run by the Product Owner.

2.3.4.2 Sprint Planning

Once the Product Planning has been finished and after a short break, it is time for the Sprint Planning. In that meeting, the team takes the user stories in the Product Backlog, in the order they are prioritized, and:

1. Decides if some items have to be subdivided and how.

2. Decides which items can be completed in that sprint, which forms the Sprint Backlog.

Moreover, for each planned item a member of the team is designated as initial owner and responsible. The reason behind this is that it allows us to make better
estimations of what the team will be able to do in the sprint (because it is easier for everybody to estimate what he can personally do, than to try estimating what the team as a whole can do) and increases the sense of commitment to finish what he said he would finish. Like we said, this is only an initial ownership and, in the middle of the sprint, the team can reassign the ownership of some tasks, depending on how the sprint is unfolding and the workload of every member of the team.

This meeting has a more technical side and therefore is run by the Scrum Master. The Product Owner’s participation is not needed, although it is welcomed. It has a timeboxing of one hour and a half.

2.3.4.3 Daily Scrum

The Daily Scrum is held every day at 10:00, except the first day of the sprint. In the 15 minutes that it lasts, all members of the team have to answer three questions:

- What have you done since last Daily Scrum?
- What will you do before the next Daily Scrum?
- What obstacles, if any, are impeding your work?

Its main goal is to reorganize and synchronize the team, and, in case someone has an impediment, gather help or ideas from other members of the team.

The starting time is arbitrary (it could be at 9am or 11am instead), the important thing is that it should be decided by consensus and then it has to be respected. In order to do that, we decided that anyone who arrives late to the Daily Scrum has to pay an euro.

2.3.4.4 Demo

The Demo is an hour long meeting held the last day of the sprint (once every two Fridays), at 12pm. In that meeting, the team shows the progress made during the sprint to the Product Owner and, ideally, to the stakeholders. It is really important to see what has been really done, find possible bugs and gather feedback in order to improve the product in next iterations.

2.3.4.5 Retrospective

The sprint officially finishes with its last meeting, the Retrospective. The Retrospective is a time for reflection, where the whole Scrum Team gathers together in a relaxed environment and talks about what happened in that sprint: what went well, what went bad and what can be improved. Using all this, the Scrum Team defines and assigns a series of actions in order to remove impediments or obstacles and another one for improving aspects of its work, generating the Impediment Backlog and the Kaizen Backlog.
It is one of the most important meetings in Scrum because it helps avoid stagnation and repetition of past errors and forces the Scrum Team to always look for possible improvements. It starts just after the Demo (once every two Fridays, at 1pm) and lasts an hour and a half.

### 2.3.5 How we estimate

#### 2.3.5.1 User stories

A user story is a high-level definition of a feature, in the customer’s language. The user stories are captured as items in Post-its, in the Product Backlog or Sprint Backlog.

The basic template of a user story is as follows:

As a <role> I want <goal/desire> so that <benefit>.

Although, in order to keep it simple, we sometimes skip the role and benefit part.

![User Story example](image.png)

Figure 2.6: User Story example.

#### 2.3.5.2 Story points

Story points are given to user stories in order to represent their abstract cost.

What do we exactly mean by abstract cost? Let’s explain it by analyzing the two words that form that term:

- We refer to it as abstract because we want everybody to agree in the perception of how much cost a task implies, regardless of how much time it takes to complete it for them. That is to say, although complete a given task could take half an hour for the CTO and complete the same task for a scholar could take two days, we want them to agree in the perceived cost of the task. What is really important is the relative difference between estimations (for everybody, a 4-point user story should take more or less twice as much as a 2-point user story), not the actual translation from points to hours.
• The word cost is used as a unit for time and difficulty. When estimating we want to consider the two of them, because there are easy tasks that are really time-consuming and tasks that do not have to necessarily take a lot of time but are difficult to solve.

When estimating, our reference is that we consider a simple screen in a mobile platform one story point.

### 2.3.5.3 Planning Poker

*Planning Poker* is a technique for estimating effort or relative size of tasks and is commonly used in agile methodologies. It uses the *Scrum Cards* seen before and the procedure that follows is the next:

1. The Product Owner, who does not participate in the estimation, gives an overview of the user story that is going to be estimated. Then, the team can ask for clarifications and discuss about risks and implications.

2. Each member of the team chooses one card representing an estimate and lays it down on the table.

3. Everyone turns his card up at the same time, revealing the estimations made.

4. If there are outliers, the person that gave the lower estimate and the one that gave the higher one try to justify their estimations. This is done because it could be that one of them has seen an easier solution or complicated parts that the rest of the team overlooked.

5. The team repeats estimation rounds until a consensus is reached.
Chapter 3

Android

This chapter talks about Android, the platform where the project at hand has been implemented. In the first section, a short overview of its history can be found and in the next section the basic components used in Android applications are presented. The intention of this chapter is to provide the reader with the bare minimum knowledge to get the feeling about how the platform works, making it easier for him to understand the rest of the thesis. The specifics of Android programming are out of the scope of this chapter.

3.1 What is Android?

In July 2005 Google acquired a small startup dedicated to mobile software development, called, in a premonitory sign, Android Inc. With this absorption, the co-founders of Android Inc., Andy Rubin, Rich Miner, Nick Sears and Chris White went to work at Google and the rumors about Google entering the mobile arena began.

Almost two years later, Android, a free, open source and Linux-based platform for mobile devices was born from the collaboration of Google and the Open Handset Alliance, an alliance of companies that included HTC, Intel, LG, Motorola and Nvidia, among many others.

The first beta version of the SDK (Software Development Kit) was released in November 2007. In September 2008, the Android SDK 1.0 was released and a month later, along with the release of the first Android device, the HTC G1, the source code of Android was released as open source, under an Apache License 2.0. Some months later, on February 2009, Android 1.1 was released.

From then on, a lot of updates, fixing bugs and adding new features, have been released, and Google has named their Android version codenames after desserts, following alphabetical order. The first major platform release, Android 1.5 Cupcake, beginning May 2009. Android 1.6 Donut on 15 September 2009. Android 2.0/2.1
Eclair in October 2009. Next, Android 2.2 Froyo on May 2010. And the last one for now, Android 2.3 Gingerbread on December 2010.

There are rumors that the two next releases are codenamed Android 2.4 Ice Cream Sandwich and Android 3.0 Honeycomb, which would violate the alphabetical order followed until now.

In any case, we can see that since its beginnings in 2005, Android has come a long, long way and, with the expected new releases and the adoption of Android by tablets and other mobile devices, we do not see limits to its growth.

3.2 Application Components

In this section, we explain the different components that provide the building blocks for developing applications in the Android platform: activities, services, broadcast receivers and content providers.

3.2.1 Activity

An activity is an Android class that presents a visual user interface to the user for one particular endeavor. It is important to note that an activity is not a view, in any case, the user interface of the activity is formed by a hierarchy of views (for example, a button, a label, an image...). If we want to use an analogy, we could think of activities as screens. For example, in an application, a screen showing the detail of a movie, where we can see its name, description, timetable, price and that has a button to buy a ticket for it, that could be an activity.

Activities are the presentation layer of the application and though they work together to form a cohesive UI, each activity is independent of the others and has an execution context on its own.

Typically, one of the activities is marked as the first one that should be presented to the user when the application is launched. In order to move between activities, the current activity can start the next one or return to the previous one.

An activity has essentially three states:

• Active or running: the activity is in the foreground and therefore is the focus for the user's actions.

• Paused: the activity has lost focus because another activity is in the foreground but it is still visible somehow (the activity in foreground is transparent, it does not cover the full screen...).

• Stopped: the activity is completely obscured by another activity laying in foreground.

As an activity transitions from state to state, it is notified of the change by callbacks to seven protected methods, that define its entire lifecycle. That can be observed in the next flowchart:
It is mandatory that all activities override at least the `onCreate(Bundle savedInstanceState)` method. The developer of the application can decide which of the other lifecycle methods wants to override, in order to respond and to do the
necessary work when the state of the activity changes. Also, all lifecycle methods should always call first its superclass version.

- void onCreate(Bundle savedInstanceState): Called when the activity is first created. All activities must override this method to do the initial setup: variable initializations, setting up the user interface, retrieving necessary data, etc. It takes a Bundle as a parameter, which contains the previous state of the activity. We can include in that Bundle any necessary information to restore the state implementing the method onSaveInstanceState(Bundle outState), which is called before killing an activity.

- void onStart(): Called before the activity becomes visible to the user.

- void onRestart(): If the activity has been stopped before, called before starting it again.

- void onResume(): Called just before the activity starts interacting with the user.

- void onPause(): Called before the system is going to resume another activity and the current activity is going into the background.

- void onStop(): Called when the activity is no longer visible to the user.

- void onDestroy(): Called before the activity is destroyed, either because the activity is finishing or because the system is destroying the activity to save space.

It is important to keep in mind that the system, in order to save space, can kill the process hosting the activity if that one is paused or stopped, or what is the same, when an activity is executing onPause(), onStop and onDestroy(). Out of this three, the only method that is guaranteed to be called before the process is killed is onPause().

For this reason and for the fact that the activity is no longer interacting with the user, onPause() is the recommended place to persist unsaved data and to stop CPU consuming processes. In any case, whatever we do in onPause() should be as quick as possible, because the next activity will not resume until the current on exits onPause().

### 3.2.2 Service

A service is a component that runs in the background for an indefinite period of time and that does not provide a visual user interface, so it does not interact directly with the user. Examples of what a service could do include downloading files, calculating something or playing music in the background.

A service has two different lifecycles, depending on how it is being used:
• The service is started and allowed to run until someone stops it or it stops itself. In order to explicitly start and stop a service the methods Context.startService() and Context.stopService() are used. Moreover, the service can stop itself using Service.stopSelf() or Service.stopSelfResult(). In any case, the calls to Context.startService() are not cumulative, so only one call to one of the stop methods is needed in order to stop the service, no matter how many times Context.startService() has been called once the service is already started.

• A client needs to interact with the service in some way. In that case, the service exposes an interface for remote objects called IBinder, written in AIDL (Android Interface Definition Language), that the client can use to communicate with it. The connection between client and service is established by calling Context.bindService() and closed by calling Context.unbindService(). Additionally, if the service was not already launched, Context.bindService() can optionally do it.

Next, a flowchart containing the two lifecycles of a service is displayed:
In the first mode (service that is running alone in background), the service has three lifecycle methods that developers can implement to monitor changes in its state:

- **void onCreate()**: Called when the service is first created. This is where the initial setup should be done.

- **void onStart(Intent intent)**: Called when the service is going to become active. This method receives the Intent object that was passed to Context.startService(), and that usually contains information needed by the service to know what it has to do.

- **void onDestroy()**: Called before the service is destroyed, either because the
service is stopping completely or because the system is destroying it in order to save space.

Moreover, if a service uses the second mode, that is to say, permits other to bind to it, apart from onCreate() and onDestroy() (onStart() is called only for services started by Context.startService()), there are additional methods to implement:

- IBinder onBind(Intent intent): Called when the client calls Context.bindService(). It receives the Intent object that was passed to Context.bindService() and if the service permits the binding, returns the IBinder interface that clients can use to interact with the service.
- boolean onUnbind(Intent intent): Called when the client calls Context.unbindService(). It receives the Intent object that was passed to Context.unbindService() and it can ask for onRebind() to be called when new clients connect to the service, returning true.
- void onRebind(Intent intent): Called for new clients connecting to the service, if onUnbind() has indicated so.

And now, the interesting part: the two modes are not mutually exclusive, it is possible to bind to a service that was started with Context.startService(). In that case, the service will execute onStart() and potentially may receive onBind() and onUnbind calls. Also, a service that has been started with Context.startService() will not completely stop until it receives one of the stop calls and all its bindings are closed.

For a complete example on using services, the Download Manager in AnkiDroid is implemented with a service that uses the two modes simultaneously and two-way communication with its clients.

### 3.2.3 Broadcast Receiver

Broadcast receivers are components that receive and react to broadcast announcements originated by the system (e.g., low battery, SD card events, user changing settings...) or initiated programmatically by an application (e.g., specific data completely downloaded to the device...), and that could be of general interest.

An application can have any number of broadcast receivers to respond to any announcements it considers important.

Broadcast receivers, like services and content providers, do not provide a visual user interface but it can affect it, for example, starting an activity after receiving an specific broadcast announcement.

Its lifecycle is formed by a single callback method:

- onReceive(Context curContext, Intent broadcastMsg): Called when a broadcast message arrives for the receiver and takes the Intent object containing the message.
Broadcast receivers are considered to be active only while executing its `onReceive()` method.

### 3.2.4 Content Provider

A content provider is a generic interface mechanism to share data between applications.

Content providers feature full permission control and are accessed using a simple URI model. Android provides some content providers for accessing to the information stored in native databases: that includes, for example, the browser, call log, contacts, media and settings. In any case, you can create your own content provider in order to expose specific data of your application to everyone.
Chapter 4

Anki

This chapter is centered in the program that inspired this project: Anki. First, we give a big picture about what Anki is and what it is used for. Next, in order to understand why Anki is so effective, an explanation about the research it is based on is presented. To finish the chapter, the most common terms used in Anki are described.

4.1 What is Anki?

Anki (which in Japanese means memorizing) is a multiplatform and open source SRS (Spaced Repetition System or Spaced Repetition Software) software developed by Damien Elmes. Anki's goal is to make memorization of any subject as easy and efficient as possible, minimizing the times the information has to be reviewed, while maximizing its retention. In order to do that, Anki uses as SRS algorithm a version of the SM-2 algorithm created by Piotr Wozniak, modified to allow priorities on cards and to show cards in order of their urgency.

Currently it has native versions for Mac OS X, Linux, Windows, iPhone, Android, Maemo, Nintendo DS, Windows Mobile (experimental support) and Zaurus, and an online version, AnkiOnline (which is not open source). At the time of starting the project, neither the Android version (that has been developed for this project) nor the iPhone one existed yet. Damien Elmes detected the necessity to cover the most powerful mobile platforms and decided to develop the iPhone version by himself, using a mobile framework known as Titanium Mobile. Currently he is working full-time on both the desktop and iOS version, and making a living of it.

It is, along with SuperMemo and Mnemosyne, one of the most important SRS in history, and nowadays, the most popular of all.
The characteristics and features that make Anki stand out over the rest are:

- Powerful spaced repetition algorithm based on SM-2, as today, the state of the art.
- Due to being multiplatform, it allows you to review anywhere.
- Synchronization features that allow you to access your latest information from everywhere.
- Shared decks to save you the hassle of creating your own decks.
- Subscriptions to other decks, so you can benefit from new additions without work on your part.
- Fully extensible through plugins, with a large number already available.
- Optimized for speed, even in really large decks.
- Supports images, audio, videos and scientific markup (LaTeX).
- Flexible and convenient way to input and edit information, based on the concept of models.
- Very active development.
- Large and active community.
- Open sourced under GPL v3.

Anki is also based in the concept of flashcards, so it allows to learn and memorize anything that can be rewritten in the form of a question-answer pair. That means that the possibilities are almost endless and that the only real limits are your time and imagination. Just as example, you could learn: languages, song lyrics, geography, people's names, poems, algorithms, definitions, phone numbers...

In terms of technology, Anki is completely developed in Python, cards are displayed in HTML (and can include, optionally, CSS and Javascript) and all the information related to decks is stored in SQLite.

The Anki Python code is divided in two modules: ankiqt for the UI part and libanki, a library that contains the algorithms for spaced repetition and synchronization, among other things.

Both can be found in Github:

- ankiqt: https://github.com/dae/ankiqt
- libanki: https://github.com/dae/libanki

### 4.2 Cognitive Psychology Concepts Used by Anki

Anki is based in two concepts or techniques from cognitive psychology, that are really simple but at the same time very powerful: active recall and spaced repetition.
4.2.1 Active Recall

Active recall is a principle of efficient learning that consists in actively stimulating memory during the learning process, as opposed to passive review methods as reading, watching or listening.

The act of recalling something strengthens the memory, and therefore is much more efficient in consolidating long-term memory.

4.2.2 Spaced Repetition

In 1885, Herman Ebbinghaus, a German psychologist that pioneered the experimental study of memory, discovered what is known as forgetting curve.

![Forgetting curve representation.](image)

The forgetting curve describes the exponential curve that illustrates how fast we tend to forget the information we learned, and Ebbinghaus determined that it could be roughly described by the next formula:

$$ R = e^{-\frac{t}{S}} $$
Where $R$ is memory retention, $S$ is the relative strength of memory and $t$ is time.

Taking a look at the forgetting curve representation we can deduce some things.

- Without repetition, any memory is doomed to be forgotten.

- The optimum time to review something is just when it is going to be forgotten (go under the 90% line). Reviewing before that point is unnecessary and inefficient because we still remember it, and waiting more in order to review it is very costly, because we are going to forget it and therefore, we will have to relearn it again.

- Each successful repetition increases the optimum interval before the next repetition is needed. That sounds pretty simple, but the consequences are remarkable. If we think about it, that means that with enough successful repetitions at the optimum reviewing time, we only have to review the specific thing we are memorizing once every many years in order to remember it. Eventually, the next optimum interval could surpass our life span, meaning that it is quite probable that we could remember it forever.

With all this, Ebbinghaus also discovered the spacing effect, that concludes that humans remember or learn more easily a subject when it is studied a few times over a long period of time (spaced presentation or spaced repetition), rather than studied repeatedly in a short period of time (massed presentation).

SRS take advantage of the spacing effect, incorporating increasing intervals of time between subsequent reviews of previously learned material.

### 4.3 Basic Terminology and Usage

#### 4.3.1 Cards

A question and answer pair is called a card. It is based on a paper flashcard with a question on one side an the answer on the back. In Anki, when the answer is shown the question is still visible by default, so that behavior is different than the one of physical flashcards. It is done on purpose because it is considered more convenient for the user, so he does not have to flip the card again if he wants to take another look at the question. In any case, this behavior can be configured.

For example, if you were studying irregular English verbs you might see a question like the next:
After thinking about the answer, you decide to go for "encoger". Once you have actively thought about your answer, you should press the "Show Answer" button.
You check that the correct answer was "encoger/se, reducir/se", so it was pretty close. It was not the exact answer but the meaning was totally correct, so you could grade your answer as "Hard" or "Good". After that, Anki will take care to show you the question at the appropriate time, in that case, 1.7 years from now if you chose "Hard" and 4 years from now if your answer was "Good".
4.3.2 Decks

A deck is a collection of cards and all its related information (statistics, facts, models...). We consider the files with extension ".anki" to be decks. The way you organize cards by decks is totally up to you: you could have a deck for every different topic or keep all your information in one deck.

![Deck files in the system file.](image)

4.3.3 Facts and Fields

A fact is a collection of related information which is then used to create cards. Facts are formed by fields, and in the example below these fields would be "Expression", "Meaning", "Image" and "Example".

![Cards in a deck.](image)
With this fact created and from its information, you could later create the next three cards:

```
shrink (shrink)
shrank (shraank)
shrunck (shrunk)
```

= encoger/se, reducir/se

“If you wash a woolen sweater with hot water, it will **shrink**.”

“As the firm had **shrunck**, they decided to lay off personnel.”

“Miraculously, her brain tumor **shrank** and finally disappeared.”

---

<table>
<thead>
<tr>
<th>Soon</th>
<th>1.7 years</th>
<th>3.7 years</th>
<th>5.1 years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Again</td>
<td>Hard</td>
<td>Good</td>
<td>Easy</td>
</tr>
</tbody>
</table>

Figure 4.6: Creating a fact.

Figure 4.7: Recognition card created from the example fact.
Figure 4.8: Picture recognition card created from the example fact.
We should note that these three cards could have been created by hand, without needing the fact concept that Anki uses. If that is so, why does Anki use facts? There are three important reasons for that:

- Efficient creation of cards. Creating all cards one by one requires a lot of repetitive typing. When we only want to generate one card there is no difference at all, but the higher number of cards that we would like to generate from the same related information the more convenient facts are.

- Efficient editing of cards. Similarly to the first point, if we made a mistake and need to edit the cards, we should have to do it one by one. If we have facts, we only have to edit the mistake on the fact and all cards generated from it are refreshed automatically.

- Better spacing between related cards. If these cards, that contain similar information, are shown closely enough we would remember the last one because we have just reviewed the other, and that is not useful. Cards from the same fact are spaced between them, so they do not appear together in a review session.

### 4.3.4 Card Models or Templates

The generation of cards from a fact is made thanks to a card model or template. A card model specifies what should appear on the front and back of a card.
If you tell Anki that you would like to generate a card from a fact and with a determinate card model, Anki will create the card and automatically fill it for you.

![Recognition card model](image)

Figure 4.10: Recognition card model.

In the picture above, we can see the card model that creates the recognition card from the fact example. In order to do that, it defines what goes in the question part and what goes to the answer part, as follows:

**Question:**
%

**Answer:**
%

<br>

<br>

<br>

<br>

4.3.5 Models

Anki allows you to store different types of information within a single deck. It separates each type of information into a separate model, that consists of a list of card models or templates and a list of fields to store in a fact.
As we can see in the next picture, the model used in the previous examples is called "English".

![Figure 4.11: English Model.](image)

That model has four fields (and for this reason the fact had also four fields): Expression, Meaning, Image and Example. And it also has three card models, that were used to generate the three cards from the example fact.
Chapter 5

AnkiDroid Analysis and Design

In this chapter, the most important aspects of the analysis and design of AnkiDroid are explained. First of all, we present the user stories from where the requirements can be distilled. Next, an explanation about the architecture where AnkiDroid fits in is given. To finish, we show the classes used in the solution and some navigational maps that display how the main functionalities will work.

5.1 User Stories

Using Scrum, a formal and exhaustive requirements analysis has not been performed. Instead of that, the usual way to take requirements in agile methodologies is through user stories, a high-level definition in the user’s language of what he wants to achieve. The main reason behind using user stories is being able to respond and adapt faster to rapidly changing requirements, while avoiding the overhead that this situation would entail with methodologies that use a more formal requirements analysis.

The user stories are created and written by the customer or a customer representative, and generally they follow the template we saw in chapter 2:

As a <role> I want <goal/desire> so that <benefit>.

Or its simplified version:

As a <role> I want <goal/desire>.

The user stories that were considered at the beginning are the following:

• As a user I want to be able to open decks created with any Anki application.
• As a user I want to be able to review my decks.
• As a user I want to be able to configure my decks.
• As a user I want to be able to configure the application’s behavior.
• As a user I want to have study sessions.
• As a user I want to be able to configure the study sessions.
• As a user I want to download my decks from AnkiOnline.
• As a user I want to upload decks to AnkiOnline.
• As a user I want to synchronize my decks.
• As a user I want to download pre-made decks from other people.
• As a user I want to see my cards as in Anki Desktop.
• As a user I want to be able to see the images in my cards.
• As a user I want to hear the audio in my cards.
• As a user I want to be able to replay the audio in my cards.
• As a user I want to be able to write my answers.
• As a user I want to be able to draw my answers.
• As a user I want AnkiDroid to be in my language.
• As a user I want to run AnkiDroid in any Android device.
• As a user I want AnkiDroid to use the device’s resources carefully.

These user stories give us a good idea of what the client wants and later, before the beginning of the sprint where the specific functionality is going to be implemented we can talk to the customer to resolve doubts, clarify the exact requirements for the task at hand at that moment and decide and write and acceptance test.

The functional requirements are obtained sprint by sprint but we can already obtain the non-functional requirements, the constraints of the user, the problem, the system and the hardware. The non-functional requirements are the following:

• work in as much Android devices as possible. Nowadays, that means that it has to support at least Android 1.5 and any screen resolution.
• present a simple and usable user-interface, considering the devices for which is being developed.
• take care of the battery consumption.
• use the network only when absolutely necessary and in an efficient way, avoiding any overload.
• take care of the RAM and CPU use.
5.2 Architecture

Figure 5.1: Complete Anki architecture at the beginning of the project.

In the picture above, we can see the global architecture that was in place before starting the project and, in a green box, how AnkiDroid fits in it.

The architecture consists in a classical client-server model, where it exists more than one client, one central server and the communication is done through Internet. The existent clients at the moment were available in the next platforms: Mac OS, Linux, Windows, Windows Mobile, Nintendo DS, Maemo and Zaurus. AnkiOnline is the web server developed by Damien Elmes that acts as a central server in all this architecture, and offers a reduced set of the functionalities of Anki. Also and most importantly, it is used for storing and synchronizing decks and it can be used as a proxy for synchronizing decks between two different clients. Other devices with platforms not listed here can still use AnkiOnline as far as they have Internet connection and a browser.
Every deck is stored in its own SQLite database, and being SQLite self-contained and relatively small, makes it very convenient for mobile platforms. For users, it is really important to have the information of their decks available everywhere they go, and that means having their SQLite databases in all their devices.

In order to do that, the SQLite databases can be taken directly from desktop and copied to the device, or the other way around. That is to say, we could take a deck created and used in Anki Desktop, copy it to the SD card of our Android device and use it with no problem. After some reviews and modifications using AnkiDroid, we could copy this deck back to our computer and open it again with Anki Desktop.

Another alternative, if the user wants to spare himself the task of moving around decks manually, is using the built-in functionality of synchronization, both in Anki Desktop and in AnkiDroid. The synchronization between clients is made using AnkiOnline as a proxy, that is to say, if we want to synchronize the changes made in AnkiDroid to Anki Desktop we would synchronize AnkiDroid with AnkiOnline and, later, AnkiOnline with Anki Desktop.

It is important to point that the synchronization is incremental and implemented in a way to transfer as little information as possible through the network, with the
objective in mind of being cost-efficient.

For all operations related with deck management, AnkiOnline exposes an API but unfortunately it is not documented. Its functioning had to be discovered investigating the Anki source code and analysing the communications with Wireshark.

### 5.3 DB Diagram

In the next page, a diagram representing the tables and their relationships in a given .anki file can be found. This diagram illustrates the complexity of the model, due to their many relationships and attributes.

The most important tables, and therefore entities, are cards, decks, facts, fields, models, fieldModels and cardModels.

The conceptual classes of the application have been derived from this DB diagram, in the correspondent iteration when they were needed. Due the fact that the majority of entities map their correspondent table, the conceptual model would be quite similar to the DB diagram.
5.4 Navigational Maps

As the whole project, the user interface has also been designed following an iterative and incremental methodology.

First of all, the navigational maps of the application have been designed, using a mockup tool. Once the navigation maps are clear, changes and modifications within every screen or view can be discussed. And finally, when that is also decided the little aesthetic details can be tweaked.

This progression is based in the fact that the better an application or view looks, the more narrow is the feedback we get. For example, if a design looks polished and pretty you will get feedback about fonts, colors and similar little details. And this is a bad thing if your user interface design has more critical flaws, like an incoherent navigation, or if you would like to explore radical different views on how to implement your higher level functionalities.

The next picture illustrates this concept perfectly:
The better it looks, the more narrow the feedback

**Looks Done**
Mocked up in Photoshop, a multimedia program (Director, Flash, etc.), or a GUI builder (NetBeans, Visual Studio, etc.)

"Can you change the font on that "I"?
Not sure I like the bevel line weight...”

**Feedback:** detailed tweaks to specific features. Very focused and incremental.

---

**Visio, Powerpoint, etc.**
Illustrated using a professional drawing or presentation tool.

“I don’t like the two-column layout for tools. Can we have them go across the top?”

**Feedback:** tweaks to the ‘screen’ or page as a whole. Incremental improvements.

---

**Rough Sketch**
Scanned from a hand-drawing, made with a drawing app and a tablet, or using the Napkin Look and Feel skin.

“Maybe the tools should be context-specific... Let’s kill the toolbar and bring up only the tools that make sense at that moment...”

**Feedback:** higher-level features are questioned, bigger changes possible.

---

**Storyboard or Use Case**
The “story” of how the user might need or want to interact with the interface (app, book, product, etc.)

“We should NOT try to put a drawing feature in here... it’s featureitis without a key benefit to most users.”

**Feedback:** big-picture ideas, possibility for revolutionary changes.

Figure 5.4: Relation between the level of detail in a design and the feedback it generates.
For all this, and for its rapid prototyping, mockups are widely used in agile methodologies in the first stages of definition of the user interface, when we want everybody to focus only on functionality.

In order to see how the different screens in AnkiDroid interact with each other, following we present to the reader the navigational maps of the most important functionalities, in a mockup form:

**5.4.1 Open deck**
5.4.2 Review deck
5.4.3 Configure preferences
5.4.4 Download deck

Menu, "Download Deck"
Choose Personal or Shared

Select deck to download
- Japanese characters
- Guitar chords
- Trivia facts
- WWII History
- Famous quotes
- Recognizing music
- Computer engineering
- Meta-AnkiDroid deck
- Math theorems

Start Reviewing  Cram

country-capitals (7 of 34 due)
Study Options
Reviews due: 122
New today: 20
New total: 29
New cards per day: 20
Session limit (minutes): 10
Session limit (questions): 200
5.4.5 Synchronize deck
5.4.6 Synchronize all decks
Chapter 6

AnkiDroid Implementation

The development of AnkiDroid has been made collaboratively between diverse developers and in different environments, but personally, has been developed in Mac OS X 10.6.4. The chosen IDE was Eclipse, in its version 3.4.2 (Ganymede) and 3.6.0 (Helios), with the ADT plugin (from ADT 0.9.4 to ADT 0.9.9).

This chapter explains how to set up the development environment and how the most significant aspects of AnkiDroid have been implemented. In case more specific and detailed information is desired, the source code can be consulted on:

https://github.com/edu-zamora/Anki-Android

6.1 Development Environment

Required:

Android SDK, Git, GitHub.

Highly recommended:

Eclipse, ADT plugin.

6.1.0.1 Android SDK

The Android SDK (Software Development Kit) provides the tools and APIs necessary to begin developing Android applications. In order to download it and install it, follow the steps on http://developer.android.com/sdk/installing.html.
After the dominance that Subversion has had in the field of VCS for many years, a new genre of VCS called DVCS (after Distribution Version Control System) is emerging with force. The main characteristic of DVCS is that each working copy contains the full history of all the revisions, thereby making it a fully functional code-base repository and a remote backup to the original code-base. That means that developers can work in his own working copy (committing, branching, etc.), without disturbing other developers nor needing internet at that moment. Later, when he is satisfied with his work he can synchronize his local repository with the remote one. The convenience that is having the full history in the working copy and the fact that is a lot easier to branch and merge in DVCS than in VCS are some of the reasons why Subversion is losing ground in front of them.

Git, Mercurial and Bazaar are the most representative DVCS.

Git was designed and developed, by Linus Torvalds, with an emphasis on speed. It is free software, licensed under the terms of the GNU General Public License v2.0. Nowadays is the most popular exponent of DVCS, because it is the most powerful, fastest and the most efficient in storage size of them. All of these but, implies that it has a more steep learning curve for beginners.

For an open source group, that potentially can have a lot of developers working at the same time, branching and merging, Git is the best choice as a VCS.

If you use Mac OS, the easiest way to get Git is downloading it from the next page: [http://code.google.com/p/git-osx-installer/downloads/list](http://code.google.com/p/git-osx-installer/downloads/list).

For more options on how installing Git, in different platforms see:

- All: Chapter 2 First Time, Section Installing Git from the book *Git Community Book* [4].

### GitHub

GitHub is a web-based hosting service for projects that use Git as revision control system. It is written using Ruby on Rails by GitHub, Inc. and it has diverse
pricing plans, but it is completely free for public repositories, what makes it ideal for open source projects.

Apart from hosting projects, GitHub also adds a social component and a lot of useful features: followers, feeds, comments in commits or even in specific lines, wikis, pages and issue tracker for individual projects, etc.

All of this makes GitHub the most popular hosting site for projects using Git.

In order to contribute code to AnkiDroid, a GitHub account is needed. You can register in https://github.com/. Once you have a GitHub account, if you want to be up to date about AnkiDroid development, follow the other usual AnkiDroid developers and don’t forget to subscribe to the news feed.

6.1.0.4 Eclipse

Eclipse is an integrated development environment (IDE), highly customizable and extensible through its plug-in system. It is mainly written in Java and it can be used to develop applications in a lot of different programming languages: Java by default and, using specific plugins, Ada, C, C++, COBOL, Perl, PHP, Python, Ruby, Ruby on Rails, Scala and Scheme. It is cross-platform (available in Linux, Mac OS X, Solaris and Windows), free and open source.

Also, one of its great advantages is the large quantity of available plugins, that allow the developer to customize it to tailor his concrete development needs: GUI development, Mobile development, RCP applications, web services, Eclipse plugins, etc.

For all these reasons, it is no surprise that Eclipse is, for any kind of development, one of the most popular IDE. In the specific case of Android development is still more popular, because Google presents it almost as the default option and because it allow us to take advantage of the ADT plugin, which simplifies and makes Android development faster.

Although it is not a requirement to use Eclipse in order to develop for Android or AnkiDroid (other IDEs like IntelliJ Idea or Netbeans can be used, or even programs like vi or emacs) we highly encourage it.

Eclipse can be downloaded from http://www.eclipse.org/downloads/. As today, we recommend to download an Helios (3.6.1) package and if you are only going to use Eclipse for AnkiDroid development, we recommend specifically the Eclipse Classic package.
6.1.0.5 ADT plugin

The Android Development Tools (ADT) plugin extends the capabilities of Eclipse to make it a powerful environment in which to build Android applications. Basically, what ADT does is provide access to the Android SDK tools from inside Eclipse. Doing that, it allows the developer to:

- Create Android projects easily using a wizard.
- Automate and simplify the building and export process.
- Add components based on the Android Framework API.
- Have a complete environment to test and debug Android applications, that includes emulation of a real device through AVDs (Android Virtual Device), virtualization of its components (like the SD card), and simulation of GPS locations, SMS and calls.
- Create in an easy and intuitive way different emulator instances or AVD (Android Virtual Device).
- Access to the logs of the applications and filter them using different criteria.
- Take screenshots.

The combination of Eclipse and ADT is the fastest way to develop for the Android platform and, although it is not mandatory, it is highly recommended to take advantage of them.

In order to install the ADT, follow the instructions on http://developer.android.com/sdk/eclipse-adt.html#installing.

6.2 The Android Manifest

All Android applications must have an AndroidManifest.xml file, placed in its root directory. This file is responsible for completely describing and specifying the application: package, components of the application (activities, services, broadcast receivers and content providers), permissions needed, minimum Android API level required, etc.

For this reason, if we want to know how AnkiDroid is implemented it makes sense to start this endeavour taking a look at its manifest (the complete Android-Manifest.xml file can be found in Appendix B).
Looking at it, we see that the structure of the application consists of twelve activities (StudyOptions, PersonalDeckPicker, SharedDeckPicker, Reviewer, MyAccount, Preferences, DeckPreferences, About, CardEditor, FactAdder, ErrorReporter, Notification), three services (CheckService, DownloadManagerService, UpdateService) and two broadcast receivers (CheckReceiver, AnkiDroidWidget).

The application tag has the attribute debuggable set to true, so AnkiDroid can be debugged when is running in a device. StudyOptions is the main activity (the first activity to be shown when the application is started), which is indicated adding the next intent to the intent-filter tag of the activity:

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

This activity also listens for the next intent:

```
<intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <category android:name="com.ankidroid.category.DECK" />
  <category android:name="android.intent.category.DEFAULT" />
  <data android:mimeType="application/vnd.anki" />
</intent-filter>
```

That intent can be used by other applications to open AnkiDroid with a specific deck. From StudyOptions we can change the configuration for the current deck, synchronize it or navigate to the principal activities in the application: Reviewer, DeckPicker, PersonalDeckPicker, SharedDeckPicker, MyAccount, Preferences, About and FactAdder.

Reviewer is where the review sessions of the user's flashcards take place. DeckPicker is a list of the local decks of the user, allowing him to load them. Similarly, PersonalDeckPicker and SharedDeckPicker are also lists of decks, but the first one is formed by decks of the user stored on AnkiOnline and the second one is formed by decks created by other Anki users. Clicking in any of these decks allows the user to download them to local, offering him the possibility to study a diverse range of subjects without the hassle of creating the decks by himself. MyAccount is a form where the user can log in with his AnkiOnline credentials, in order to be able to download his decks stored there and to synchronize decks from AnkiDroid. Preferences is a set of options to configure and personalize the way the application works. The About activity explains briefly to the user what is and what does AnkiDroid and how he can get in touch with us, the developers and creators. CardEditor and FactAdder allow the user to edit individual cards and add facts, respectively. And finally, ErrorReporter allows the user to send us a report after a crash has happened.
DownloadManagerService is the service responsible to manage all downloads and to notify the interested activities when something happens.

The service UpdateService and the broadcast receiver AnkiDroidWidget are used by the widget.

And last, Notification, CheckService and CheckReceiver, from the Veecheck library, are used to notify users when there is a new version of the application.

The minimum API level required, the permissions and the type of screens supported are placed after closing the application tag. In its goal to reach as much people as possible, AnkiDroid supports from API level 3 (or Android 1.5, the oldest Android version still used) onward and supports all kind of screens. The permissions needed are vibrate, receive a notification when the system boot is completed, internet and access to check the network state.

6.3  User Interface

The most important activity of AnkiDroid, Reviewer, is formed by a WebView, buttons, text labels, a text editable field and a Whiteboard, a custom class that allows the user to write in the screen.
In terms of UI, we should also highlight the three activities in the application that have lists, DeckPicker, PersonalDeckPicker and SharedDeckPicker. These activities are interesting because they have the particularity that make use of an adapter, a bridging class that binds the underlying data to its representation in the UI. DeckPicker uses one of the standard Android adapters, SimpleAdapter. The other two activities are even more interesting because they implement their own custom adapter. This custom adapter displays different information according to the type of the element and creates the list headers that separates them. Also, is a good example on how to apply the Holder View pattern. In the next picture, we can see how a list with the custom adapter looks like:
The layout of the other activities in the application are formed by more common elements: buttons, text labels, text editable fields, images, WebView, etc.

6.4 SRS

In order to explain how SRS works, it can be divided in its two most important parts:

- Getting the next card to review.
- Answering the card, and updating its fields and statistics accordingly.

6.4.1 Getting the next card to review

The SRS is only used in the Reviewer Activity. When the user navigates to this Activity, its onCreate method is called, where the preferences are restored, the member variables and the user interface are initialized and the first card to show is asked for. The latter is done with the next instruction:
DeckTask is a class that extends AsyncTask<DeckTask.TaskData, DeckTask.TaskData, DeckTask.TaskData> and it is used for all time-consuming tasks related to handling a deck. In Android, the expensive operations have to be run in a different thread from the main thread, in order to avoid blocking the UI. Although that can be done with typical Java threads, it is cleaner and more convenient for the programmer to use AsyncTask instead.

The instruction seen above takes three parameters: the first one is the kind of task to perform, the second one is the listener used to interact with the UI and the last one is a DeckTask.TaskData object, a wrapper of all the info needed to perform the operations (ease, deck and card) and also used to return back to the Activity results generated.

In the doInBackground method for this specific task, doInBackgroundAnswerCard, happens most part of the work:

```java
if (oldCard != null) {
    deck.answerCard(oldCard, ease);
}
newCard = deck.getCARD();
publishProgress(new TaskData(newCard()));
```

The card passed to DeckTask, if exists, is answered with the given ease and, right after that, the next card is retrieved. When the DeckTask.launchDeckTask method is called from onCreate, a null value is passed as the card parameter, therefore no card is answered and the DeckTask only returns the next card to display, which is the desired behavior because the reviewing session is just being started.

The card is returned to the UI through the publishProgress method. That method calls automatically to the onProgressUpdate method of the AsyncTask, which at the same time calls to the onProgressUpdate method of the listener mAnswerCardHandler. This method takes as a parameter a TaskData object, which in this case is the new card to display.

```java
@Override
protected void onProgressUpdate(TaskData... values) {
    mListener.onProgressUpdate(values);
}
```
But what does the method `deck.getCard()` exactly do? How the SRS decides which card is the next one that has to be reviewed? In order to explain that, but keeping it as simple as possible for the sake of readability and understanding, the process of how the next card is picked is represented in a flowchart form:
Figure 6.3: Get next card flowchart.
The `getCard` method on `Deck.java` chooses a card to review from failed cards, review cards or new cards, depending on its availability and on the deck's configuration. If no card is available for review now, a `null` is returned and `Reviewer` finishes the session.

All that completes the process required to get the first card of a session. With this card `Reviewer Activity` updates the UI accordingly and the user can start reviewing cards for this session.

### 6.4.2 Answering a card

In the middle of a session, when the user clicks in one of the four ease buttons (buttons to grade how well he knew the answer to the question displayed), the process to answer the card is triggered. The four ease buttons have associated a callback method called `onClick`, where the variable `mCurrentEase` is updated and the method `DeckTask.launchDeckTask` is called again, this time as follows:

```java
DeckTask.launchDeckTask(
    DeckTask.TASK_TYPE_ANSWER_CARD,
    mAnswerCardHandler,
    new DeckTask.TaskData{
        mCurrentEase,
        AnkiDroidApp.deck(),
        mCurrentCard
    }
);
```

Now, a card not null is passed like parameter, so this time the instruction `deck.answerCard(oldCard, ease)` on `doInBackgroundAnswerCard` is executed. In it, the attributes and statistics of the answered card are updated, the time gap (space) until the card is due for review again is calculated, the spacing of any related cards, if any, is also updated and a new entry in the reviewing history is created. The next flowchart shows how this process works:
Figure 6.4: Answer card flowchart.
After the card has been answered, the doInBackgroundAnswerCard method continues as has been seen before, getting the next card to review and passing it to the UI through its listener.

## 6.5 Download Manager

The Download Manager has been implemented in DownloadManagerService, a class that extends Service. It is responsible for downloading personal decks and shared decks, and for updating shared decks. In order to do that, it implements three AsyncTask: DownloadPersonalDeckTask, DownloadSharedDeckTask and UpdateDeckTask. Also, it allows the user to pause or cancel updates, but not downloads yet due to a problem with the server, that does not implement correctly the Range attribute and therefore does not allow downloads to be restarted. Anyway, the code in DownloadManagerService is prepared to support these features once the problem with Range is solved.

The Service is designed with the purpose of running the exact time needed to complete its tasks, even surpassing the life of the Activities in the application. Once it has finished all its work, it stops itself.

It exposes an AIDL interface (Android Interface Definition Language), IDownloadManagerService, to allow Activities to bind to it and then communicate with it. We can see what the Service allows to do to the Activities in the interface definition:

```java
interface IDownloadManagerService {
    void registerPersonalDeckCallback(IPersonalDeckServiceCallback cb);
    void unregisterPersonalDeckCallback(IPersonalDeckServiceCallback cb);
    void registerSharedDeckCallback(ISharedDeckServiceCallback cb);
    void unregisterSharedDeckCallback(ISharedDeckServiceCallback cb);
    List<Download> getPersonalDeckDownloads();
    List<SharedDeckDownload> getSharedDeckDownloads();
    void downloadFile(in Download download);
    void resumeDownloadUpdating(in Download download);
}
```

For their part, if the Activities want to be notified when there is some change in the state of a download they have to implement the correspondent AIDL interface (IPersonalDeckServiceCallback or ISharedDeckServiceCallback) and register this interface to the Service (with either registerPersonalDeckCallback or registerSharedDeckCallback). Doing that, the Service is able to call back to all its observers when something of interest happens.
6.6 Synchronization

Before being able to synchronize, the user has to go to MyAccount Activity and log in with his AnkiOnline username and password. If the log in is correct, the username and password are saved in Preferences, because they are going to be needed in the synchronization calls to the server. This is done by calling to the following method:

```java
private void saveUserInformation(String username, String password) {
    SharedPreferences preferences = PrefSettings.getSharedPreferences(getActivity());
    Editor editor = preferences.edit();
    editor.putString("username", username);
    editor.putString("password", password);
    editor.commit();
}
```

If the user is not logged in and tries to synchronize, an alert explaining this situation is shown and it directs the user to MyAccount Activity.

Once logged in, a user can start the synchronization process in two different ways:

- In StudyOptions Activity, selecting Menu and clicking on the menu option Sync.
- In DeckPicker Activity, clicking on the Sync all button.

The instruction to start the synchronizing process from StudyOptions is the next one:

```java
Connection.syncDeck(
    syncListener,
    new Connection.Payload(
        new Object[] { 
            username,
            password,
            deck,
            mDeckFilename
        }
    )
);
```
And in DeckPicker is used the following one:

```java
Connection.syncAllDecks(
    mSyncAllDecksListener,
    new Connection.Payload(
        new Object[] { username, password, mDeckList }
    )
);
```

As we can see, the two calls are similar. They both use Connection, a class that extends `AsyncTask<Connection.Payload, Object, Connection.Payload>` and that is used for all time-consuming tasks that require internet connection. All public methods in Connection take two parameters: the first one is a listener that is used to communicate with the UI and the second one is a `Connection.Payload` object, a useful wrapper to pass any data (observe that `Connection.Payload` is initialized with an `Object[]` array) to the task and to retrieve the results.

`Connection.syncAllDecks` receives a list of decks and its `doInBackground` method, `doInBackgroundSyncAllDecks`, calls to the `doInBackground` method associated with `Connection.syncDeck`, `doInBackgroundSyncDeck`. This last method receives all data needed to synchronize a deck: username and password of the user, the deck to synchronize and its complete path.

The process followed on `doInBackgroundSyncDeck` is represented in the next flowchart:
Figure 6.5: Synchronization flowchart.
In order to fully understand how and why the synchronization process works, we should explain what the concepts of summary and payload mean.

A summary is a set formed with all entities created, modified or deleted since the last synchronization. For each one of these entities, only the identifier and the time of creation, modification or deletion is stored in the summary, in order to be efficient (the server summary is sent to the client and it would be highly inefficient to send the complete info for each entity if we don’t need it). Both summaries are used later in the process to determine if a full synchronization is needed (if a lot has changed it is more efficient to do a full synchronization than to apply the changes one by one) and in the generation of the payload.

The payload generated in the client side is a list created from the comparison of the two summaries, that explains what has happened from the point of view of the client: what has been added, with the complete data, what has been deleted and what is missing. The server receives this payload and acts accordingly, creating and deleting the indicated entities and returning the complete info of the ones that were missing in the client.

6.7 i18n

In order to reach as much users as possible, it is really important for any application to be internationalized, at least to the most used languages and ideally to all languages with potential users.

So far, there has been work translating AnkiDroid from English to nineteen other languages: Catalan, Chinese Simplified, Chinese Traditional, Czech, Danish, Dutch, Finnish, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Polish, Portuguese, Romanian, Spanish and Swedish.

How all these translations have been done and managed is explained in more detail in chapter 7. In this chapter the focus is on what has to be done at implementation level in order to support i18n.

Fortunately, Android makes that process very simple.

All text strings used in the application for the default language should be written in res/values/strings.xml. If additional languages are desired, a new values resource with a qualifier specifying the language or language-region combination has to be created. This new values resource has to contain a strings.xml file, where all text strings are translated to the specified language.

For example, our res/values/strings.xml contains all text strings for English (our default language), in the form of key=value.

```xml
<string name="deckpicker_loaddeck">Loading deck... </string>
```
If we wanted to translate this text to Catalan, we should create a values resource for Catalan (res/values-ca) and place the correspondent translation in res/values-ca/strings.xml.

```xml
<string name="deckpicker_loaddeck">Carregant baralla...<\string>
```

Later, in order to access the text strings from code the class Resources has to be imported and the key of the text string wanted passed to its getString method, like in the next example:

```java
import android.content.res/Resources;
...
Resources resources = getResources();
String i18nText = res.getString(R.string.deckpicker_loaddeck);
```

And depending on the current Locale Android loads the text string indicated by the key deckpicker_loaddeck from res/values/strings.xml or from res/values-ca/strings.xml. If the specified key is not in the correspondent strings.xml file according to the Locale, Android falls back and tries to load it from res/values/strings.xml. If it is not even there, then the application crashes.
Figure 6.6: Example of English text.

Figure 6.7: Example of Catalan text.
Chapter 7

AnkiDroid Open Source Community

When talking about a project, us computer scientists tend to focus only on the engineering part (taking requirements, designing a good enough solution for the problem at hand, testing it, getting feedback and repeating the cycle) and sometimes forget that an open source project involves a lot more than just coding or the software development process.

This chapter serves as a reminder of that situation, explaining what the general goal for open source projects is and what tools, infrastructure and documentation is needed in order to increase the chances to achieve it, survive and be successful. Once presented all that in a general way, we show how it applies to AnkiDroid.

7.1 What Does an Open Source Project Need?

OK, so you have an idea for an open source project. Now what? Should you start coding right away? If you want the project to succeed and grow a healthy and active community around it then the answer is no, not yet.

First of all, you should look around to see if there is already an open source project that does what you want. If you find some that it is close to what you want, it makes more sense to join that project and add the functionalities that are missing than starting a complete project from scratch. There is no need to reinvent the wheel, unless you want to start the project as an educational and learning experience. Even in the case that your ideas for the project don’t match with what the people on it want to do, you always can take the code and do with it whatever you want.

In case you don’t find any project or you want to start your own project anyway, for whatever reason, we explain in the next subsections what you need to do and have for your new project. It is also important to know the goal we are trying to achieve with all this: assure that we clearly communicate what the project exactly
does or intends to do, get it into the hands of the right people, make sure that those
who receive it know how to use it and entice as many people as possible to get
involved in the ongoing and never-ending task of improving the project.

Don’t wait until the project is completely finished to set them up or you are going
to struggle way more than it is necessary and miss the point of doing a project in an
open source way. But on the other hand, you don’t have to do them all before you
start really implementing the project. The answer is do them as soon as possible
and as soon as it makes sense.

7.1.1 People

The key element of any open source project, so much that we dedicate it a sub-
section, is the people that interact with it. It is really important to stress that the
main goals for an open source project is acquire users and acquire contributors,
and every tool and documentation that we explain in this section is directed toward
this objective.

How do you get users? Getting users is a two step process, first, your project
has to provide enough value to be interesting to users and, second, you have to let
them know about the existence of your project and how they can get it.

How do you convert users into contributors? All boils down to treating every
user as a potential contributor, letting them know how they can contribute, sharing
credit and responsibilities and letting them feel that they make a difference, because
they really do, and that it is also their project, because it really is.

The chances are that if your project is worth paying attention to, you cannot do
it alone, so recruit like-minded people and let them help you build your idea. That
is what open source is about.

7.1.2 Website

The website of the project is the first page users are likely going to see, so it should
give them a welcoming overview of the project and it should point them to the rest
of the documentation or tools associated with the project. Its goal is to give a first
good impression, let the users quickly know what the project is about and, if they
are interested, show them where they can find the rest.

7.1.3 License

If you want to release your project and make it open source, you need a license
for it. Unlicensed code is implicitly copyrighted and therefore anyone that wants to
use it has to explicitly ask for permission first. Later, we explain how to choose and
apply a license, but for now keep in mind that you need one.

Also, state clearly in the front page of your website that the project is open
source and free, indicating which license it uses. It costs nothing and not doing so
can cause the loss of many potential developers and users.
7.1.4 Features and Requirements List

You should have a list of the features your project provides and another one stating the requirements needed in order to run it, if any. The goal is not wasting the time of the people evaluating your project, so they can quickly know if it is what they are looking for, they can run it or they want to get involved.

7.1.5 Version Control

For many reasons, the use of a Version Control System is indispensable. First of all, the obvious thing: a VCS allows developers to manage code changes conveniently and keep all files versioned, in case something happens. But in open source projects is also vital because it allows anyone to have real-time access to the latest development version and because it enables everyone to watch directly what is happening to the code.

7.1.6 Issue Tracker

An Issue Tracker is a useful tool that stores one of the most important kind of information, information about how to improve the project. It enables everyone to see the state of the development and to file new bugs, tasks and feature requests. Also, enables developers to keep track of what they are working on, coordinate with each other and plan releases.

Now, is one of the times that having a lot of users pays off. More people using the application and more eyes looking at the code usually means a greater quantity of feedback on the Issue Tracker, which is key to improve.

To the visitors’ eyes, it also represents how active a project is, and paradoxically having a lot of bugs there gives a good impression. The actual quantity of bugs depends on three factors: the absolute number of bugs present in the software, the number of users using it and how convenient is register them into the Issue Tracker. Every project has an indeterminate number of bugs awaiting to be discovered so the really important is how aware the community is about its shortcomings, how much involved are the users and how well it registers and prioritizes those bugs.

If your Issue Tracker is empty because your project is just starting, make that clear somewhere (the front page is a good place) so the visitors don’t confuse a young project with a dead one.

7.1.7 Communication channels

Any project needs some common space where users, developers and other contributors can meet, connect and talk. That forms the communication channels of a group. It is a good decision to have available at least one persistent and low interrupting communication channel (like mailing lists or forums) and at least one interactive and in real time (like IRC channels or chat rooms).
Visitors usually want to know how to reach the people involved with the project somehow, so state clearly the addresses of mailing lists, forums, chat rooms and IRC channels that the community uses.

Also, we have to take into account that the use and management of communication channels has to evolve according to the number of people using them. For example, at the beginning of a project is fine to have only a mailing list for both users and developers. Users will probably be happy to know how the development is going but that is until their inbox gets flooded by one hundred emails per day. Once the number of developers and mails that they write to each other in order to communicate is high enough, is a smart decision to create a new mailing list exclusively for developers. The point is, acknowledge the communication necessities of your project and, taking into account the current volume of communications, choose the most appropriate channels for it.

7.1.8 Documentation

All open source projects need two types of documentation: one addressed to users and another one to developers.

The user documentation should consist of a short explanation about how to install the application, an overview of how it works, a guide explaining how to perform common tasks and a FAQ (Frequently Asked Questions).

The developer documentation should help potential developers to quickly learn how everything is done, so they can start working on the project as soon as possible. It can include guides explaining which environment is used, how to set up the project, which code style guidelines are applied, guidelines on using the VCS, the Issue Tracker, etc.

The documentation is going to improve and evolve through time but it is important that there is something as soon as possible, even if it is rudimentary and incomplete.

7.1.9 Translation strategy

If the project is expected to work on different languages, you have to think about how the translations are going to be done. Are you going to hire translators? Are you going to translate it all by yourself? Are you going to let contributors take care of the translations? And, depending on what your answer is, you might need to look for a tool that helps you managing the translation work.

7.2 AnkiDroid Community Infrastructure

In this section we explain what tools have been adopted, what documentation has been created and how the infrastructure of the AnkiDroid community looks like.
AnkiDroid is still a relatively young project, so its documentation and processes are still far from perfect, but they are going to improve given enough time.

The first big thing to decide is where to host the major part of our activity. It exists the possibility to pick a specific service or tool for each of the things we listed before or even implement some of them yourself, but we consider best to keep everything as centralized as possible and choose a canned hosting site, that usually offers various of them.

In our case we chose to use Google Code, that provides revision control, an issue tracker, a wiki for documentation and a file download feature. Google Code is free for open source projects licensed under Apache, Artistic, BSD, GPLv2, GPLv3, LGPL, MIT, MPL or EPL, so it is free for AnkiDroid due to its GPLv3 license. Also, it works with any Google account, which makes it very convenient because al-
most anyone has one, so it is not necessary to register for a new service, and because simplifies the communication between contributors. AnkiDroid is using Google Code to host its website, user documentation, FAQ and part of the developer documentation (part of it, the Code Conventions, are included as annex), to manage its activity through the issue tracker and to allow users download files related to the project. The only service that is not being used is the hosting of the source code. The reason for this is that Google Code only offers Subversion and Mercurial as VCS, and none of them were the selected one.

Instead, Git was chosen as a VCS for its speed, efficiency and power. The source code then is hosted in GitHub, the most popular hosting service for Git, and with a reason: apart from the hosting, it offers a series of useful features like following developers, watching repositories and feeds for different activity, which makes it very convenient to stay up to date with what the other developers on AnkiDroid are working on.

Regarding communication channels, the AnkiDroid project has three mailing lists and one IRC channel. The three mailing lists are provided by Google Groups, which was selected again for the convenience of using Google accounts, for its easy but powerful configuration options and for its great spam prevention.

The first and most important mailing list is our forum, used by both users and developers alike. So, in the AnkiDroid mailing list we can find help requests, ideas, suggestions for improvements and technical discussions. Although the group has grown a lot since the beginning, the quantity of mails still does not justify separating the user and developers communications in different mailing lists.

The second mailing list, AnkiDroid bugs, receives the mails sent by the error reporter integrated in the AnkiDroid application. We have implemented a service that assist us identifying the bugs with more occurrences from the mailing list and helps us associating these bugs with issues in the issue tracker. This mailing list contains private information and it is not addressed to users, so only the managers of AnkiDroid have access to it.

The third and last mailing list, AnkiDroid commits, it is exclusively addressed to developers interested in AnkiDroid development. Its goal is to have an space to practice code review and discuss commits of interest. Until now it has not been used enough, but code review is a key practice to foster productivity and quality, so we have to promote its use.

For a more interactive communication, in real-time, at first we relied exclusively on GTalk. It is not a bad solution and we still use it sometimes, but it has two inconveniences:

• It has too many distractions: mails to read, friends, family, etc.
• By default, it does not encourage talking in group.

For these reasons, we decided to create an IRC channel on Freenode, the largest free and open source software-focused IRC network.
Finally, we needed a way to handle translations. We don’t have the resources to hire translators neither we have the necessary expertise to do it all by ourselves, so we rely on contributors to translate AnkiDroid to as much languages as possible. For this endeavour we chose Crowdin, a collaborative translation tool that supports multiple formats (GNU gettext, iPhone/MacOS X .strings, Qt .ts, Android strings.xml, Blackberry .rcc, Symbian, Microsoft .NET .Resx, Java .properties, Flex .properties, DKLang), is community friendly and includes an online editor with suggested translations and a translation API. Crowdin is still in beta and for now is completely free to use, but seems that in the future they intend to charge for their services, except to open source projects.

All this completes the overview of the infrastructure that we created and use everyday working on AnkiDroid. Some workflow and documentation has still to be polished and improved but the basic infrastructure is already set up and working.

The addresses of all the services presented are:

- AnkiDroid bugs mailing list: Not provided because it is not intended for users.
- AnkiDroid commits mailing list: http://groups.google.com/group/ankidroid-commits.
- IRC channel: irc.freenode.net#ankidroid.
- Crowdin page: http://crowdin.net/project/ankidroidv0-5.

### 7.3 Licensing

#### 7.3.1 Choosing a License

##### 7.3.1.1 Options

At the time of choosing a license for an open source project two main options exist: create a new license or use a preexisting one. Although creating your own license has the advantage of being able to tailor it to exactly cover the needs and characteristics that your project requires, using an existing license is better for two main reasons:

- Familiarity: If you use one of the more popular open source licenses, a lot of people will be already familiar with it and they will not need to read the whole license again in order to use your code.
- Quality: The existing licenses have a very solid base in terms of legislation, product of good lawyers and much thought and experience.
Obviously, we don’t have neither the recourses or the expertise needed to create a license exclusively for AnkiDroid, so we chose to stick with one of the already existent open source licenses.

7.3.1.2 Principal Open Source Licenses

In the software world there are a great variety of licenses and a complete list of them can be found on http://www.gnu.org/licenses/license-list.html. Here, we only consider the following open source licenses, for being the most popular ones: GNU General Public License v3 (GPL v3.0), GNU Lesser General Public License v3 (LGPL v3.0), BSD License, MIT License and Apache v2.0.

All these licenses agree on the basics:

- A user can obtain a program’s source code.
- A user can modify the source code.
- A user can redistribute the source code, both in original and modified form.
- Copyright holders and authors provide no warranties whatsoever.

The differences between them are represented in the next table [5, 6]:
<table>
<thead>
<tr>
<th>Can you release commercial works?</th>
<th>GPL v3.0</th>
<th>LGPL v3.0</th>
<th>BSD</th>
<th>MIT (X11)</th>
<th>Apache v2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can you create derivative work?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Compatible with proprietary software?</td>
<td>No</td>
<td>Yes (the software that links to the library is not considered a derivative work)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Compatible with GNU GPL?</td>
<td>Yes</td>
<td>Yes</td>
<td>Only the modified BSD license (the original BSD license enforces an advertising clause)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Comments</td>
<td>The GPL dominates the free software world but its viral component can potentially scare clients.</td>
<td>Not viral like its GPL counterpart. Software can be dynamically linked to other LGPL licensed libraries without having to release your source code under LGPL. This license is generally used for software libraries.</td>
<td>Popular because its flexibility. There are really no limitations to what the licensee can do with the software other than the attribution requirements.</td>
<td>Popular because of the extreme simplicity of its text. The whole license is about half a page long and is very permissive like the BSD license.</td>
<td>Similar to the BSD license, but it goes into further detail in the attribution clauses and maintenance of intellectual property rights. Choosing this license over the BSD or MIT license is a matter of how specific you want your protections to be.</td>
</tr>
</tbody>
</table>

Table 7.1: Open source licenses comparison.
From all these licenses, GPL v3.0 was the chosen one for licensing AnkiDroid. The main reasons that led us to take that decision are:

- Recognizability of the license: GPL v3.0 is by far the license most used in the world, so a lot of the developers that will interact with the code at some point are already familiar with it, so they won’t have to spend extra time to read and understand the license.

- Enforcement of a free license on derivative works: We are developing AnkiDroid in an open source way for the benefit of users and developers alike and we want that any improvement made to the code is also available for everybody.

- Consistency with Anki Desktop: Anki Desktop also uses a GPL v3.0 license and it is good to keep consistency within the Anki framework, whenever it makes sense.

### 7.3.2 Applying a GPL v3.0 License

Once the license has been chosen, it has to be applied to the project and stated in the project’s front page. It is not necessary to put the complete text of the license there, only the name is enough.

In order to apply the license to the project, the next steps have to be followed:

1. Put the full license text in a file called COPYING (or LICENSE)

2. Put a short notice at the top of each source file, stating the copyright date, holder, license and where the complete text for that license can be found.
   
The short notice is valid as far as it contains all this information, but it usually follows the next format:

   ```
   Copyright (c) 2009 Edu Zamora <edu.zasu@gmail.com>

   This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

   This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

   You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.
   ```
Chapter 8
Planning and Economic Analysis

This chapter presents the different management aspects of the project. In the first section, an explanation about how the project has been planned and managed is given while in the second one, an economic analysis is supplied.

8.1 Timeline

In the next page, a Gantt chart showing the work done for the project can be found. It should be said that the Gantt chart has been simplified and only includes the more important and interesting things I personally did in the project. Also, a blue line representing a task suggests that the work on that given task was ongoing and in progress at the time, but it does not mean that I was working on it a hundred percent of my time. We can see that there are more than one task per day, so that would be impossible. The reason behind all these simplifications is that it would be impossible to represent all the tasks done in a single sheet, with exact precision, and still be understandable by the reader, so in favor of clarity the chart has been kept as simple as possible.

Once the project and the main goals for it were chosen, we set a timebox of a year in order to accomplish them. Given the amount of work to do, try to finish this project in less than one semester was unrealistic, two semesters could be a little risky and three semesters gave us one year to do it, and some additional flexibility accounting for unforeseen events.

The time finally needed to complete the entire project has been almost 15 months (from the last days of October 2009 to January end), taking into account that I was working at the same time. This time was divided in four main stages: Training, Synchronization, Download Manager and Documentation.

The complete planning has not been made beforehand because we are using an agile methodology like Scrum. Instead of this, a planning of two weeks work
has been made at the beginning of each sprint, allowing us to take into account the real available time for that sprint, my energy at the time and the work other people was doing, if any. Also, the monitoring of the development of the project has been conducted every two weeks, in the Scrum demo and retrospective meeting. In addition, the thesis of the project was started in the last days of September but the most important part of it has been written in the last two months.
1) Preparation and Training
   1.1) Preparing the environment
   1.2) Android Platform

2) AnkiDroid Development
   2.1) Media support
      • 2.1.1) Audio support
      • 2.1.2) Replay audio
      • 2.1.3) Image support
      • 2.1.4) Auto-scale images
   2.2) Synchronization
      • 2.2.1) Requirements analysis
      • 2.2.2) Design
      • 2.2.3) Synchronization implementation
      • 2.2.4) Integration
      • 2.2.5) User feedback
      • 2.2.6) Synchronize all decks
      • 2.2.7) Testing
   2.3) Download Manager
      • 2.3.1) Basic download
      • 2.3.2) Design
      • 2.3.3) Login and UI redesign
      • 2.3.4) Multi-database access
      • 2.3.5) Service Implementation
      • 2.3.6) Integration
      • 2.4) Automatic testing
      • 2.5) Minor tasks/bugs

3) Open Source Community
   • 3.1) Licensing
   • 3.2) Documentation
      • 3.2.1) User documentation
      • 3.2.2) Developer documentation
   • 3.3) User support

4) Project Documentation
   • 4.1) Thesis of the Project
   • 4.2) Project Report
   • 4.3) Preparing the oral defense
8.2 Economic Analysis

During the development of the project different kinds of tasks have been performed, that in reality belong to specific professions. So, in order to give a more accurate estimation, we distribute the hours worked in the next job positions:

- **Java/Android developer**: It is responsible for all tasks related with software engineering: choosing the best technologies and tools for each specific problem, taking requirements, designing the solution, implementing it and writing the necessary technical documentation. On the other hand, it has not to perform the testing by himself, that is left for the testers.

- **Tester**: It is responsible for designing, preparing and executing a battery of tests in order to assure the correctness of the code.

- **Customer service professional**: It is responsible for communicating with users of the application, resolving their doubts, answering their questions and helping them in whatever he can.

- **Translator**: It is responsible for translating the texts in the application and in the documentation to other languages.

First of all, an estimation of my participation’s cost in the project is given, and, later, an estimation of the total cost of the project is presented. There are two reasons to make this separation in the estimation of costs: the first one is the difference in precision and accuracy. While I tracked almost all hours I worked on the project in order to be able to make an accurate estimation, a lot of assumptions and bold guesses had to be made to estimate the total cost. The second reason, is that doing that allows us to see the incidence of my work in the project.

The cost per hour for each profession has been calculated taking the median salary for this position from mysalary.com and making the necessary operations, assuming a conversion ratio from dollar to euro of 1:0.7623.

We have to take into account that these costs per hour are based on median North American salaries, so the calculated total costs are completely different than they would be if we used salaries from another country. For that matter, the total cost of the project is substantially higher than it would be in Spain. We did not take Spanish salaries as a reference because this is an international project and therefore people of different countries are working on it. Due to the impossibility to choose salaries that hold true for every country, we took, in an arbitrary decision, North American salaries as a reference.
Table 8.1: Cost of my work in the AnkiDroid project.

The total cost of the project was calculated with the help of three members of the AnkiDroid community: taking their time estimations for each kind of task, calculating the mean time spent on them, assuming six AnkiDroid contributors based on the mean values and adding these hours to mine.

Table 8.2: Total AnkiDroid project cost.
Chapter 9

Conclusions

9.1 Evaluation of the Goals

At this point, we can affirm that the goals set for the project have been achieved:

- Android, Scrum and the open source movement have been studied, understood and, more importantly, applied in order to implement the project.

- We set the basic infrastructure in place for the AnkiDroid open source community, that has gone from inexistent to become a large and active community in that time.

- We were successful in developing a version of AnkiDroid that was suitable for reviewing. That included the design of the complete user interface, the implementation of the SRS algorithm, the synchronization algorithm, a download manager for all the downloads, loading decks, configure study options, configure preferences, manage your AnkiOnline account, an about page, media management (audio and images), correct visualization of flashcards, implementation of various answering methods (typing the answers or writing them with the whiteboard), DB management and testing. Additionally, we went beyond what the scope set for that project was, thanks to the participation of a few developers in the last months, something that was impossible to foresee while planning it. So, now we even have the possibility to add, edit, mark and suspend cards, to delete decks and to choose between different study methods.

- An open and close relationship with our users, and with the other contributors for that matter, has been formed. That can be seen in the high participation in our forums (that counts with 218 users and around 1500 messages), in our issue tracker (more than 300 issues reported) and the number of downloads of our application (more than 4000 in our project page and more than 20000 in Android Market, where it has 433 reviews with an average rating of 4/5 stars). Personally, this project has increased immensely my volume of
communication, with more than 300 mails related to AnkiDroid written and in-
numerable hours spent reading and analysing the communication generated
by users and other contributors.

9.2 Future Work

Next, the most important functionalities or tasks to complete in order to keep im-
proving the AnkiDroid project are listed:

• Finish the ongoing adaptation of the code to libanki1.1, an update of the
Python library made by Damien Elmes a few weeks ago. This update has
affected the SRS and synchronization algorithm, so in order to be able to use
the same decks in Anki Desktop, AnkiOnline and AnkiDroid the adaptation
has to be completed.

• Improve and automatize our testing process as much as possible, because in
an open source project where anyone can commit to the repository it is vital
to maintain the quality of the code. In order to achieve that, my intention is to
implant as much as possible TDD (Test Driven Development), design and set
in place a complete and automatized battery of tests, with the help of Maven,
Robotium and Roboelectric, and adopt CI (Continuous Integration) practices.

• Add support of the complete set of Anki functionalities to AnkiOnline.

• Add a social aspect and improved sharing functionalities to AnkiOnline.

• Complete and improve the available documentation and its organization.

• Adopt a Release early, release often philosophy when it comes down to the
releases to Android Market, and set the necessary processes to support it.

• Improve the graphic design aspect of AnkiDroid, something that was out of
the scope of this project.

• Add statistics to AnkiDroid.

• Add video support to AnkiDroid.

• Add text-to-speech functionality to AnkiDroid.

• Complete internationalization of AnkiDroid in as much languages as we can.

9.3 Reflections on SRS

First of all, I want to express my thoughts about SRS after all my experience with it.

In my opinion, it is an incredible study tool that I am sure will revolutionize the
education world, once it goes mainstream, and will be an integral part of institutions
like high schools and universities. The promise of SRS algorithms is that, if you take
a few minutes to review every day, you are going to eventually remember forever everything you input into them, with less repetitions and with a retention rate of at least +90%. And in my experience, they deliver.

If you are not impressed with that, I encourage you to try a good SRS for a month, and see for yourself what achieving a retention rate of +90% in 5-10 minutes per day feels like. I am sure you will feel that you learned more than usual, due to the decreased repetitions, and remember a lot of it... and you even could feel that these short reviewing sessions are quite fun (or at least, more fun than waiting for someone, for the bus or for whatever reason while doing nothing).

So, if I think SRS are so great why are they not more popular? In spite of the many good things SRS have, I am aware that not everything about them is perfect and, in my humble opinion, there are some really important aspects they have to improve before the general public completely adopts them:

- Learning curve and usability: SRS applications usually introduce a lot of new terminology and they have a lot of options to configure, and at the beginning, new users don’t have any idea what they mean or what they intend to do. I would say that currently SRS are more geared toward power-users (and that includes Anki Desktop and AnkiDroid) and if we want more people to adopt them, we will have to lower the barriers to entry. Simplicity will be key.

- Easier input and easier collaboration: One of the most annoying things about SRS is getting the information you want to memorize. There are two possible paths for that: you either create your own decks from scratch or you download pre-made decks. Usually, the first option takes a lot of time but results in a high quality deck that fulfills your unique learning necessities, while the second option saves you a lot of time and work but introduces a lot of information you do not want or need to learn. A solution for this problem would be designing a way to reduce the work at the time of inputting information or/and creating an space where people can collaborate in the creation of high quality decks and cards, and share and search them easily. I am thinking about a kind of collaborative social web dedicated to flashcards.

- Marketing: I do not know how, but we have to change the way we sell (and by sell I do not mean charge, I mean pitch the idea... that is an open source project after all) the concept of SRS. Maybe, the selection of a word or acronym more memorable and that resonates better with people than SRS would do the trick.

9.4 Personal Conclusions

On a more personal note, and in a year that has been tough for me, this project has been an enormous but at the same time wonderful challenge, that has made me learn, grow, face some of my flaws and fears and overcome the doubts and struggles I encountered along the way.
Looking backwards, I am very proud of how all this unfolded. I see myself prior starting the project, with no real working experience, with any knowledge about Android, Scrum or open source at all and, back then, it was difficult for me to imagine how I could organize and manage an open source project, with all its consequences and responsibilities.

But I did. And in the process I had the opportunity to meet wonderful people from all around the world and learn about mobile and Android development, Python, Git, testing practices, LaTeX, agile methodologies, Scrum and how to work with users and contributors.

Additionally, the fact of working in an international open source project pushed me to completely do it in English. English has been a passion of mine for the last two years and my level was not bad, but I had never written more than a few emails in it. And with that experience under my belt, the prospect of writing a complete thesis in English was truly intimidating, to say the least. There is no doubt though, that with more than a hundred pages and more than three hundred emails written for that project, my English level has skyrocketed.

On the other hand, participating in an open source project not only provides you with a lot of learning opportunities, it also gives you a great feeling of contribution, both with the development community and with users. I wanted my project to be more than just an academic project, that dies when the course is over, and with open source what we are doing is legacy: it will go on, evolve and serve as an example even when we, the original creators, are not there anymore. Regarding users, when some of them tell you that what you are doing has really improved his life, you can only feel moved.

All in all, I recommend to any computer engineer to work in an open source project of his own interest, because it gives you everything we need at our jobs: knowledge, experience, ability for working in a team, capacity to communicate properly and even some job offer.

But said that, we have to be aware that not every aspect of working in an open source project is positive, specially in the context of a PFC. In my opinion, there are three aspects that are less convenient in an open source project than in a personal project, and that have made my PFC more difficult.

First, it increases brutally the time spent on communication. As the number of people on a team increases, the number of communication paths increases too, and in a multiplicative way. And the more communication paths you have, the more time you spend communicating and the more opportunities to create some misunderstanding. In a project on your own the time spent on communication is almost negligible, so you can fully dedicate yourself to the different phases of the software engineering process.
Second, the decisions you make have to be consistent with everything that surrounds the project. In our case, that was Anki and a lot of the decisions of design were made to be consistent with it. On the other hand, when you are creating a project from scratch, you can decide the most convenient solution for every problem without any worry.

Finally, it introduces a lot of risk and uncertainty. It is impossible to plan long-term and to know for sure how everything is going to unfold because you are not the only one that has a say on it. For a person like me, that likes to have everything under control and has some perfectionist tendencies, Scrum has been a great help here. Scrum has allowed me to plan every two weeks and not feel overwhelmed about what could happen in the distant future, while reacting and adapting to changes.

After all this, you realize what is the exact role of university in our education. University does not provide us with everything we need... but it provides us with something more crucial and important. It teaches us the basic concepts, practices, methodologies and skill sets that allows us to survive, learn and improve in our own. And that is all we need.

With those last sentences, I am writing the end of my PFC "AnkiDroid: Open Source Development in Android" and, as a consequence, the end of my stay at university as a computer engineering student. Fortunately though, this is not the end of my work in AnkiDroid.
After a deserved time off, I will come back to working in the trenches, side by side with my colleagues, to playing and learning, and to giving away what we create for the world to enjoy.
Appendix A

AnkiDroid Manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ichi2.anki"
    android:versionName="0.5 alpha8" android:versionCode="13"
    android:installLocation="auto">
    <application android:label="@string/app_name" android:icon="@drawable/anki" android:name="AnkiDroidApp" android:debuggable="true">
        <activity android:name="StudyOptions">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="com.ankidroid.category.DECK" />
                <data android:mimeType="application/vnd.anki" />
            </intent-filter>
        </activity>
        <activity android:name="DeckPicker" android:label="@string/deckpicker_title" />
        <activity android:name="PersonalDeckPicker" android:label="@string/personaldeckpicker_title" />
        <activity android:name="SharedDeckPicker" android:label="@string/shareddeckpicker_title" />
        <activity android:name="Reviewer" android:label="Reviewer" android:configChanges="keyboardHidden|orientation" />
        <activity android:name="MyAccount" android:label="My Account" />
        <activity android:name="Preferences" android:label="@string/preferences_title" />
        <activity android:name="DeckPreferences" android:label="@string/deckpreferences_title" />
        <activity android:name="About" android:label="@string/about_title" />
        <activity android:name="CardEditor" android:label="@string/cardeditor_title" />
        <activity android:name="FactAdder" android:label="@string/factadder_title" />
        <activity android:name="ErrorReporter" android:label="Error Reporter" />
        <receiver android:name="com.ichi2.veecheck.Notification" android:label="@string/notifications_title">
            <intent-filter>
                <action android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```
<action
    android:name="android.intent.action.BOOT_COMPLETED"/>
</action>
</intent-filter>
<action
    android:name="com.ichi2.anki.VEECHECK_RESCHEDULE_CHECKS"/>
</action>
</intent-filter>
<action
    android:name="com.ichi2.anki.VEECHECK_CONSIDER_CHECK"/>
</action>
</intent-filter>
</receiver>
<service android:name="com.ichi2.veecheck.CheckService">

</service>
<service android:name="com.ichi2.anki.services.DownloadManagerService"/>
</service>
<br/– Broadcast Receiver that will process AppWidget updates —>
<receiver android:name=".AnkiDroidWidget" android:label="AnkiDroid">

</receiver>
<br/– Service to perform web API queries —>
<service android:name=".AnkiDroidWidget$UpdateService" />
</application>
<uses-sdk android:minSdkVersion="3"/>
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<supports-screens
    android:largeScreens="true"
    android:smallScreens="true"
    android:anyDensity="true"/>
</manifest>
Appendix B

AnkiDroid Code Conventions

B.1 Introduction

This appendix lists the recommendations applied to AnkiDroid, an Android project based on Anki.

If you find any error on this appendix or any inconsistency between the recommendations presented here and the automatic checkstyle and formatter for Eclipse, please send an email to ankidroid@gmail.com.

B.1.1 Layout of the recommendations

Layout for the recommendations is as follows:

<table>
<thead>
<tr>
<th>n. Guideline short description</th>
<th>Example if applicable</th>
</tr>
</thead>
</table>

B.1.2 Recommendations importance

In this appendix the words must, should and can are used to convey the importance of a recommendation. A must requirement has to be followed, a should is a strong recommendation and a can is a general suggestion.

B.1.3 Automatic style checking and formatting

In order to benefit from automatic code style checking, following the recommendations below, install the Checkstyle plugin in your IDE and use the configuration file ankidroid.checkstyle.xml.

In Eclipse, if you would like to format automatically the code, download the configuration files ankidroid.cleanup.xml, ankidroid.formatter.xml and
and import them into Eclipse. All these files can be found in the folder `docs/eclipse` of the project.

B.2 General recommendations

1. Any violation to the guide is allowed if it enhances one of the following, by order of importance (higher levels cannot be sacrificed for lower levels):
   • Logical Structure
   • Consistency
   • Readability
   • Ease of modifications

B.3 Naming conventions

B.3.1 General naming conventions

2. All names must be written in English.

3. Package names must be in all lower case.

   ```java
   com.company.application.ui
   com.sun.eng
   edu.cmu.cs.bovik.cheese
   ```

4. Class names must be nouns, in mixed case with the first letter of each internal word capitalized.

   ```java
   class Line;
   class AudioSystem;
   ```

5. Variable names must be in mixed case starting with lower case.

   ```java
   int age;
   float availableWidth;
   ```
6. Constant (final variables) names must be all uppercase using underscore to separate words.

```java
public static final int SOME_CONSTANT = 42;
```

7. Negated boolean variable names must be avoided.

```java
boolean isLoaded;
boolean isError;
```

8. Method names must be verbs in mixed case, with the first letter in lowercase and with the first letter of each internal word capitalized.

```java
getName();
computeTotalWidth();
```

9. The terms `get/set` must be used where a class attribute is accessed directly.

```java
author.getName();
author.setName(name);
point.getX();
point.setX(3);
```

10. Non-public, non-static field names should start with `m`.

```java
private long mLongVariable;
private int mIntVariable;
```

11. Static field names should start with `s`.

```java
private static MyClass sSingleton;
```

12. Associated constants (final variables) should be prefixed by a common type name.

```java
public static final int COLOR_RED = 1;
public static final int COLOR_GREEN = 2;
public static final int COLOR_BLUE = 3;
```

13. Functions (methods with a return) should be named after what they return and procedures (void methods) after what they do.

14. In a method name, the name of the object is implicit and should be avoided.

```java
employee.getName();
```
15. Abbreviations in names should be avoided.

computeAverage();
ActionEvent event;

16. Abbreviations and acronyms should not be uppercase when used in a name.

getCustomerId(); // NOT: getCustomerID();
exportHtmlSource(); // NOT: exportHTMLSource();

17. Plural form should be used on names representing a collection of objects.

ArrayList downloads;
int[] points;

18. num prefix should be used for variables representing a number of objects.

int numPoints = points.length();

19. Number suffix should be used for variables representing an entity number.

int employeeNumber;
int comicNumber;

20. Exception classes should have Exception like a suffix.

class CustomException extends Exception {
    ...
}

21. Singleton classes should return their unique instance through the getInstance method.

class MySingletonClass {
    private final static MySingletonClass sInstance = new MySingletonClass();
    private MySingletonClass() {
        ...
    }
    public static MySingletonClass getInstance()
        return sInstance;
}
// NOT: get() or instance()...

22. Generic variables should have the same name as their type.

void setView(View view);
// NOT: void setView(View v); void setView(View aView);
void close(Database database);
// NOT: void close(Database db); void close(Database sqliteDB);
23. Variables with a large scope should have very descriptive (and usually long) names, variables with a small scope can have short names.

24. is prefix should be used for boolean variables and methods. Alternatively and if it fits better, has, can and should prefixes can be used.

```java
boolean isVisible;
boolean isOpen();
boolean hasLicense();
boolean canEvaluate();
boolean shouldAbort = false;
```

25. The term compute can be used in methods where something is computed.

```java
valueSet.computeAverage();
matrix.computeInverse();
```

26. The term find can be used in methods where something is looked up.

```java
vertex.findNearestVertex();
matrix.findSmallestElement();
node.findShortestPath(Node destinationNode);
```

27. The term initialize can be used where an object or a concept is set up.

```java
initializeViews();
```

28. Iterator variables can be called i, j, k, m, n...

```java
for (int i = 0; i < downloads.size(); i++) {
    statements;
}
```

B.3.2 Specific Naming Conventions

29. String key resources must be all lowercase using underscore to separate words.

```xml
<string name="good_example_key">Example value</string>
<--! NOT:
<string name="badExampleKey">Example value</string>
```

30. XML elements identifiers must be all lowercase using underscore to separate words.

```xml
<TextView android:id="@id/good_id_example" ... 
<!— NOT: <TextView android:id="@id/badIdExample" ... -->
```
31. Test method names should be composed by a name representing what is being tested and a name stating which specific case is being tested, separated with underscore.

```java
void testIsDistinguishable_protanopia() {
    ColorMatcher colorMatcher = new ColorMatcher(PROTANOPIA);
    assertFalse(colorMatcher.isDistinguishable(Color.RED, Color.BLACK));
    assertTrue(colorMatcher.isDistinguishable(Color.X, Color.Y));
}
```

32. Activity names can have Activity like a suffix.

```java
public class ExampleActivity extends Activity {
    ...
}
```

B.4 Layout techniques

B.4.1 Length line

33. File content should be kept within 120 columns.

B.4.2 Indentation

34. Basic indentation should be 4 spaces, without using tabs.

```java
if (condition) {
    statements;
    ...
}
```

35. Indentation of any wrapping line should be 8 spaces, without using tabs.

```java
if ((condition1 && condition2)
    || (condition3 && condition4)
    ||!(condition5 && condition6)) {
    doSomethingAboutIt();
}
```
B.4.3 Braces

36. 1TBS (One True Brace Style) must be used. That means:

- Opening brace "{" appears at the end of the same line as the declaration statement.
- Ending brace "}" takes up an entire line by itself and it is intended at the same level that its correspondent opening statement.
- Braces are mandatory, even for single-statements or empty blocks.

```java
class MyClass {
    int func() {
        if (something) {
            ...
        } else if (somethingElse) {
            ...
        } else {
            ...
        }
    }
    // NOT:
    // if (condition) body();
    // if (condition)
    // body();
}
```

B.4.4 White spaces

37. White space should be used in the following cases:

- After and before operators.
- Before an opening brace.
- After Java reserved words.
- After commas.
- After semicolons in for statements.
- After any comment identifier.

```java
a = (b + c) * d; // NOT: a=(b+c)*d
if (true) {} // NOT: if (true){
while (true) {} // NOT: while(true) {
doSomething(a, b, c, d); // NOT: doSomething(a,b,c,d):
for (i = 0; i < 10; i++) {} // NOT: for(i=0;i<10;i++){
// This is a comment // NOT: //This is a comment
```
B.4.5 Blank lines

38. Three blank lines should be used in the following circumstances:
   • Between sections of a source file.
   • Between class and interface definitions.

39. Two blank lines should be used between methods.

40. One blank line should be used in the following circumstances:
   • Before a block or single-line comment.
   • Between logical sections inside a method, to improve readability.

B.5 Control structures

B.5.1 if

41. The if-else class of statements must have the following form:

   if (condition) {
       statements;
   } else if (condition) {
       statements;
   } else {
       statements;
   }

B.5.2 for

42. The for statement must have the following form:

   for (initialization; condition; update) {
       statements;
   }
B.5.3 while

43. The while statement must have the following form:

```
while (condition) {
    statements;
}
```

B.5.4 do-while

44. The do-while statement must have the following form:

```
do {
    statements;
} while (condition);
```

B.5.5 switch

45. The switch statement must have the following form:

```
switch (condition) {
    case ABC:
        statements;
        // falls through

    case DEF:
        statements;
        break;

    default:
        statements;
        break;
}
```
B.5.6  try-catch

46. A try-catch statement must have the following form:

```java
try {
    statements;
} catch (Exception exception) {
    statements;
}

try {
    statements;
} catch (Exception exception) {
    statements;
} finally {
    statements;
}
```

B.6  Comments

47. All comments must be written in English.

48. Commented-out code must be deleted.

49. TODO and FIXME must be written all in capitals and followed by a colon.

```plaintext
// TODO: Good TODO       // TODO -> Bad TODO
// FIXME: Good FIXME     // fixme: Bad FIXME
```

50. The code should be self-documenting and therefore the use of comments should be minimized as much as possible.

51. Comments should be used for complex algorithms and for explaining non-obvious decisions.

52. Comments should not be used to compensate for or explain bad code. Tricky or bad code should be rewritten.
53. There should be a white space after any comment identifier.

```java
// Good comment    // Bad comment
/**
 * Good comment  * Bad comment
 */
```

54. Comments should be indented relative to their position in the code.

```java
// Good comment
something();
// Bad comment
something();
```

55. The TODO comment should be used to indicate pending tasks, code that is temporary, a short-term solution or good enough but not perfect code.

56. The FIXME comment should be used to flag something that is bogus and broken.

57. Javadoc comments should have the following form:

```java
/**
 * Return lateral location of the specified position.
 * If the position is unset, NaN is returned.
 *
 * @param filePath Complete name of the file.
 * @return File.
 * @throws IOException If an I/O error occurred
 */
public File openFile(String filePath) throws IOException{
    ...
}
```

58. // should be used for all non-Javadoc comments, including multi-line comments.

```java
// Comment spanning
// more than one line.
```

59. All public classes and all public and protected methods within public classes should be documented using Javadoc conventions.
60. If a collection of objects can not be qualified with its type, it should be followed by a comment stating the type of its elements.

```java
private Vector points; // of Point
private Set shapes; // of Shape
```

61. For separation comments within different parts of a file, the following forms should be used (depending on its level of importance):

```java
// ********************************
//
// ********************************
// _____________________________
//
// _____________________________
```

### B.7 Logging

62. All logs must be written in English.

63. Logs must never contain private information or protected content.

64. `System.out.println()` or `printf` (in native code) must never be used.

65. The use of logs in release should be strictly restricted to the necessary ones.

66. Logs should be terse but still understandable.

67. The **ERROR** level should only be used when something fatal has happened.

68. The **WARNING** level should be used when something serious and unexpected happened.

69. The **INFORMATIVE** level should be used to note something interesting to most people.
70. The `DEBUG` level should be used to note what is happening on the device that could be relevant to investigate and debug unexpected behaviours.

71. The `VERBOSE` level should be used for everything else.

72. A `DEBUG` log should be inside an if (LOCAL_LOGD) block.

```java
if (LOCAL_LOGD) {
    Log.d(TAG, "Debugging application");
}
```

73. A `VERBOSE` log should be inside an if (LOCAL_LOGV) block.

```java
if (LOCAL_LOGV) {
    Log.v(TAG, "Informing about current state");
}
```

B.8 File organization

74. The `package` statement must be the first statement of the file.

75. The import statements must follow the package statement. import statements should be sorted by importance, grouped together by packages and leaving one blank line between groups. The ordering of packages according to their importance is as follows:

- Android imports
- Third parties imports (com, junit, net, org)
- Java imports (java and javax)

Within each group, the order of imports is alphabetical, considering that capital letters come before lower letters (e.g. Z before a).

```java
import android.widget.TextView;
import android.widget.ToggleButton;
import com.ichi2.utils.DiffEngine;
import com.tomgibara.android.veecheck.util.PrefSettings;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
```
76. Imported classes should always be listed explicitly.

```java
import android.app.Activity; // NOT: import android.app.*;
```

77. Class and Interface declarations should be organized in the following manner:

1. Class/Interface documentation.
2. `class` or `interface` statement.
3. Class (static) variables in the order `public`, `protected`, package (no access modifier), `private`.
4. Instance variables in the order `public`, `protected`, package (no access modifier), `private`.
5. Constructors.
7. Inner classes.

78. Android Components (Activity, Service, BroadcastReceiver and ContentProvider) declarations should be organized in the following manner:

1. Component documentation.
2. `class` statement.
3. Class (static) variables in the order `public`, `protected`, package (no access modifier), `private`.
4. Instance variables in the order `public`, `protected`, package (no access modifier), `private`.
5. Constructors.
6. Lifecycle methods (ordered following the natural lifecycle, from creation to destruction).
7. Android methods.
8. Other methods.
9. Inner classes.
79. Methods should be vertically ordered following the next two criteria:
   • Dependency: If one function calls another, they should be vertically close, and the caller should be above the callee, if at all possible.
   • Conceptual Affinity: Methods that perform similar tasks or have similar naming should be vertically close.

80. Method modifiers should be given in the following order:
   • access modifier: public, protected or private
   • abstract
   • static
   • final
   • transient
   • volatile
   • synchronized
   • native
   • strictfp

```java
public static double square(double a); // NOT: static public double square(double a);
```

B.9 Miscellaneous

B.9.1 General

81. Static variables or methods must always be accessed through the class name and never through an instance variable.

```java
AClass.classMethod(); // NOT: anObject.classMethod();
```

82. The incompleteness of split lines must be made obvious.

```java
method(param1, param2,
    param3);
setText("Long line split" +
    " into two parts.");
```

83. Special characters like TAB and page break must be avoided.
84. Each declaration should take up an entire line.

```java
int level; // NOT: int level, size;
int size;
```

85. Each statement should take up an entire line.

```java
i++; // NOT: i++; j++;
j++;
```

B.9.2 Types

86. Type conversions must always be done explicitly. Never rely on implicit type conversion.

```java
intValue = (int) floatValue;
// NOT: intValue = floatValue;
```

87. Arrays should be declared with their brackets next to the type.

```java
int[] points = new int[20];
// NOT: int points[] = new int[20];
```

B.9.3 Variables and constants

88. Variables must never have dual meaning.

89. Variables should be initialized where they are declared and they should be declared in the smallest scope possible that makes sense.

90. Floating point variables should always be written with decimal point and at least one decimal.

```java
double total = 0.0; // NOT: double total = 0;
```

91. Floating point variables should always be written with a digit before the decimal point.

```java
double probability = 0.5;
// NOT: double probability = .5;
```

92. Numerical constants (except, in some cases, -1, 0 and 1) should not be coded directly. Use constants instead.

```java
private static final int TEAM_SIZE = 11;
...
Player[] players = new Player[TEAM_SIZE];
// NOT: Player[] players = new Player[11];
```
B.9.4 Operators

93. Embedded assignments must be avoided.

\[
a = b + c; \quad \text{// NOT: } d = (a = b + c) + r;
\]

\[
d = a + r;
\]

94. The assignment operator should not be used in a place where it can be easily confused with the equality operator.

\[
// \text{ NOT: } \text{if} \ (c++ = d++) \{ \ldots \}
\]

95. Parenthesis should be used liberally in expressions involving mixed operators in order to make the precedence clear.

\[
\text{if } \ ((a == b) \&\& (c == d)) \quad \text{// NOT: } \text{if} \ (a == b \&\& c == d)
\]

96. If an expression containing a binary operator appears before the ? in the ternary ?: operator, it should be parenthesized.

\[
(x >= 0) ? x : -x; \quad \text{// NOT: } x >= 0 ? x : -x;
\]

B.9.5 Conditionals

97. Complex conditional expressions must be avoided. Introduce temporary boolean variables instead.

```java
bool isFinished = (elementNo < 0) || (elementNo > maxElement);
bool isRepeatedEntry = (elementNo == lastElement);
if (isFinished || isRepeatedEntry) { ...'

// NOT:
// if ((elementNo < 0) || (elementNo > maxElement) || elementNo <=
// lastElement) { ...
```

98. In an if statement, the normal case should be put in the if-part and the exception in the else-part

99. Executable statements in conditionals should be avoided.

```java
InputStream stream = File.open(fileName, "w");
if (stream != null) { ...'

// NOT:
// if (File.open(fileName, "w") != null)) { ...
```
### B.9.6 Loops

100. Only loop control statements must be included in the `for()` construction.

101. Loop variables should be initialized immediately before the loop.

102. `do-while` loops can be avoided.

103. The use of `break` and `continue` in loops should be avoided.

### B.9.7 Methods

104. Methods should be as short as possible.

105. Methods should perform one and only one thing.

### B.9.8 Exceptions and finalizers

106. Exceptions must not be ignored without a good explanation.

107. Generic Exception should not be caught except at the root of the stack.

108. Finalizers should be avoided.
B.9.9  Android AsyncTask

109. Overriden methods in an AsyncTask should be organized in the following order:

- onPreExecute
- onProgressUpdate
- onPostExecute

B.9.10  Android XML Layouts

110. LinearLayout elements must explicitly state the orientation attribute.

```xml
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
>
<!−− NOT
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
>
    −−>
```
111. At the same level of the element's name, it should only appear the `xmlns` attribute or the `id` attribute, in this order.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/example_id"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
</RelativeLayout>
```

```
<RelativeLayout android:id="@+id/example_id"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
</RelativeLayout>
```

```
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
</RelativeLayout>
```

---

```
<RelativeLayout android:id="@+id/example_id"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
</RelativeLayout>
```

---

112. Each attribute should take up an entire line.

```
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
</LinearLayout>
```

```
<LinearLayout
    android:layout_width="fill_parent" android:layout_height="wrap_content">
</LinearLayout>
```

---
Attributes should be grouped by type (layout properties, text properties...) and within the same type, they should be ordered alphabetically.

```xml
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/example_id"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#ffffff"
        android:textSize="24sp"
        />
</LinearLayout>
```

---

```xml
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/example_id"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#ffffff"
        android:textSize="24sp"
        />
</LinearLayout>
```

---

```xml
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/example_id"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#ffffff"
        android:textSize="24sp"
        />
</LinearLayout>
```

---
B.10 Cheat sheet

This section intends to summarize in a brief and visual way the most important guidelines and recommendations stated in the rest of this appendix, helping the developers working on AnkiDroid to quickly grasp or remember the most essential aspects of style used in the project.
<table>
<thead>
<tr>
<th>What</th>
<th>Naming Conventions</th>
<th>Good examples</th>
<th>Bad examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>lower case</td>
<td>com.company.application.ui</td>
<td>com.company.Application.Ui</td>
</tr>
<tr>
<td>Class</td>
<td>nouns, mixed case, starting with uppercase</td>
<td>AudioSystem</td>
<td>audioSystem</td>
</tr>
<tr>
<td>Variable</td>
<td>mixed case, starting in lowercase</td>
<td>availableWidth</td>
<td>AvailableWidth, available_width</td>
</tr>
<tr>
<td>Non-public, non-static field</td>
<td>mixed case, starting with ( m )</td>
<td>mLongVariable</td>
<td>longVariable, LongVariable</td>
</tr>
<tr>
<td>Static field</td>
<td>mixed case, starting with ( s )</td>
<td>sSingleton</td>
<td>singleton, staticVariable</td>
</tr>
<tr>
<td>Constant</td>
<td>uppercase, using underscore to separate words</td>
<td>SOME_CONSTANT</td>
<td>some_constant</td>
</tr>
<tr>
<td>Method</td>
<td>verbs, mixed case, starting with lowercase</td>
<td>getName()</td>
<td>get_name()</td>
</tr>
<tr>
<td>String key</td>
<td>lowercase, using underscore to separate words</td>
<td>good_example_key</td>
<td>badExampleKey</td>
</tr>
<tr>
<td>XML element identifier</td>
<td>lowercase, using underscore to separate words</td>
<td>good_id_example</td>
<td>badIdExample</td>
</tr>
</tbody>
</table>

Table B.1: AnkiDroid naming conventions.
<table>
<thead>
<tr>
<th>Line length</th>
<th>120 characters</th>
</tr>
</thead>
</table>
| Indentation | Normal: 4 spaces with no tabs  
Wrapping lines: 8 spaces with no tabs |
| Braces style | 1TBS braces style: class declarations, method declarations, block statements |
| White spaces | Before and after: operators  
Before: opening brace  
After: Java reserved words, commas, semicolons in for statements, any comment identifier |
| Blank lines | 3 blank lines:  
Between sections of a source file  
Between class and interface definitions  
2 blank lines:  
Between methods  
1 blank line:  
Between local variables in a method and its first statement  
Before a block or single-line comment  
Between logical sections inside a method |
| File organization | Copyright/ID comment  
package declaration  
import statements |
| Class organization | Class/Interface documentation  
class or interface statement  
Class (static) variables in the order public, protected, package (no access modifier), private  
Instance variables in the order public, protected, package (no access modifier), private  
Constructors  
Methods  
Inner classes |
| Android components organization | Component documentation  
class statement  
Class (static) variables in the order public, protected, package (no access modifier), private  
Instance variables in the order public, protected, package (no access modifier), private  
Constructors  
Lifecycle methods (ordered following the natural lifecycle, from creation to destruction)  
Android methods  
Other methods  
Inner classes |
| Order of imports | Android imports, third parties imports (alphabetically), Java imports (java and javax) |
| Order of method modifiers | public protected private abstract static final transient volatile synchronized native strictfp |

Table B.2: AnkiDroid code guidelines summary.
Bibliography


