Project Plan

Master Practicum "Android OS", WS 2012/13

Zaur Molotnikov, Bibek Shretsha
zaur.molotnikov/bibek.shretsha@in.tum.de

November 10, 2012

Abstract

This document is a rough project plan, a part of the Android OS Master Practicum, MSc Informatics program at TUM. In the document we describe the idea of an upgrade for the AnkiDroid software, which lie in the basis of the project as well as try to come up with time estimations on each of the parts. We shortly discuss, why we think the ideas are challenging and important, together with criteria or goals we put on the resulting work. We mention here the desired process model, we are going to take in order to reach the goals.

Contents

1 Description of the project

1.1 Anki software

AnkiDroid is an Android user interface (UI) for the bigger Anki project. The project contains desktop cross-platform application, web-application and services, as well as mobile client or UI.

The whole goal of the Anki software is enabling the user to learn new information, by memorizing cards. The software, however is different from many memory-cards applications in some ways. First, it has an algorithm to track, how well the user is remembering the information, advising on which cards has to be repeated on a particular date.

The cards have usually two sides, and thus represent an entity in the data base with two most-meaningful fields. The user composes sets of the cards, which are called decks. And then has chance to review the composed decks. The program allows manipulation with decks, such as: generating new cards from existing ones (for example, through reverting), sharing decks on the web server, exchanging decks with other Anki users, synchronizing decks (and the usage info) among different Anki clients a single user may run.

Potential usages for the learning application with cards could be, Anki (b):

1. learning a language
2. studying for medical and law exams
3. memorizing people's names and faces
4. brushing up on geography
5. practicing guitar chords
6. mastering long poems

plus, what we identified is could also be possible:

1. learning pronunciation
2. learning music types
3. learning chemistry formulas
4. teaching children
5. developing visual memory

Figure 1: AnkiDroid Screenshot : Imported from PC version graphics, is not possible to input in AnkiDroid

The full capability of all learning techniques can only be achieved by a powerful enough cards application, which would support different sorts of content on the cards. For example, to study formulas (either in chemistry, or mathematics) the graphical content would be much more appropriate, than textual. To memorize people's names and faces, as the project creator suggest, also photos are essential. Learning a foreign language put other demands on the application: it has to support pronunciation of complicated words, (like "Wiedervereinigung", German for reuniting, or "Dostoprimechatelnosti", Russian for sights), as well as different fonts, to show Arabic spelling or hieroglyphs.

1.2 AnkiDroid and its limitations

Displaying of content as described in a previous subsection is a must for the application. Additionaly, on a modern mobile device, the creation of such content could be especially productive, due to high interactivity level with the user. One could imagine recording voice from the microphone, capturing photos on the camera, or drawing illustrations with the sensor screen capabilities.

Unfortunately, such mentioned features are currently missing from the Android implementation of Anki, but are required by community, and are constantly asked for Anki (a)

The existing implementation of AnkiDroid contains only raw textual input. Simply adding a better WYSIWYG editor for the text to have styles there, would be a great improvement. However this is not the main target for us.

Often while learning something the user starts from something he/she does not know. This causes a need to search for knowledge. In a context of the Android application for cards, it would mean many steps process to finish one cards:

1. unknown information found (for example, in a browser)
2. new deck or simply new cards started in Anki
3. search for the meaning started (browser, photo application, etc.)
4. needed meaning found
5. second side of the card filled in back in Anki

The required application switch process decreases user productivity. Moreover, due to the nature of Android activities life cycle, it could be, that within such switching, due to the lost of a current state, the user will need to repeat the work twice (creating cards, after unfinished was discarded, for example). This motivates adding to the application itself information providing capabilities. Among such capabilities we consider the usage of embedded recording and capturing devices, as well as search on the available on the Web resources for information (for example, Google images sarch, Beolingus dictionary, Wikipedia, etc.)

Summarizing, there is a broad spectrum of opportunities towards improving in a meaningful and requested by community way the application. Trying to implement some of the mentioned features for the application represents the goal for the project.

1.3 Real-world project

It is worth mentioning, that dislike many existing student toy-applications, the modification of AnkiDroid represents a real world project. The application for the Android under question is not only extremely popular (more than 100.000 downloads), but is also very reliable, and has clear above average (4.5+) rating. Application represents a client for Android, however it must well integrate with all other existing clients as well as the web service, provided by Anki project, which enables, for example deck exchange.

Thus development features on top of such application:

1. is a challenging task for student level programmers
2. requires certain level of quality during development

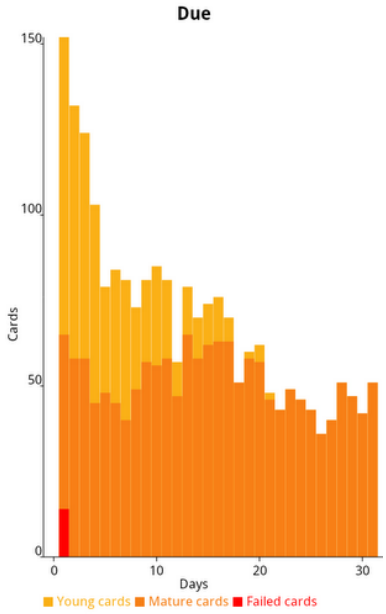

Figure 2: AnkiDroid Screenshot : Displaying learning progress

3. requires maturity of each feature for publishing
4. allows for real-world project experience
5. puts consideration on the visual integrity with existing software
6. introduces need to research on the existing code base, and
7. brings up the need in considering and validating integration

2 List of tasks

2.1 Approach to handle time and estimation

Here we present the concrete list of functions, we want to implement during the project run. We also provide rough time estimation, however, it should be noticed, that since we are dealing here with a complex project, and each feature requires complete integration before completion, which can cause additional, not planned tasks (re-factoring, existing bugs fixing, etc.), it is not possible to precisely predict the amount of time, we will have to spend on accomplishment. That is why we present an open list of tasks, which we see now, and we will try to accomplish as much as possible during the project run, and within the dedicated time slot. After the project time is over, we will present more detailed per-hour report on what has been done,

Figure 3: AnkiDroid Screenshot : Loading existing deck from a web service.

bound to the artifacts, so it will be possible to track, what has been really done.

Here we give the estimation from below, and whenever it is impossible on the current stage, to predict, how much time a certain step is going to require, we use question mark sign. If a sub-task with unknown time appears N times, the parent task in the time estimation is going to have + N x ?.

During the discussion of potential features to implement with the original developer (Nicolas Raoul), we got also approximate time estimation, which we show here in square brackets. The approximation by original author are "if all goes well" positive and were made without considering detailed implementation issues we can face. However, they are used to "validate" our estimations.

2.2 Features to implement

1. Existing code base learning, development preparation: 18 + 1 x ? hrs.
   - Getting the code for AnkiDroid run : 3 hrs.
   - Analyzing the program structure, documentation: 5 hrs.
   - Learning third-party libraries, used in the program: 2 hrs.
   - Analyzing existing web services and the way they are used: 5 hrs.
   - Analyzing the desktop client and interoperability requirements: 3 hrs.

- Setting up the web services substitution for development purposes: ? hrs.

2. Adding picture to a card (from another app, intents): 13 + 2 x ? hrs. [40 hrs.]

   - Analyze, how the additional information can be stored with web-services: 1 hr.
   - Analyze, if the reformatting of picture (shrinking) is needed and the way to do it: 1 hr.
   - Implementing the reformatting of the image: ? hrs.
   - Extending UI to allow picture insertion: 3 hrs.
   - Integrating with the web services: ? hrs.
   - Validation of functionality and testing: 4 hrs.
   - Code-review, last refactorings and release: 4 hrs.

3. Adding sound to a card (from another app, intents): 14 + 2 x ? hrs. [40 hrs.]

   - Analyze, how the additional information can be stored with web-services: 1 hr.
   - Analyze, if certain compression algorithm/audio format has to be used: 2 hr.
   - Implementing the compression/reformatting: ? hrs.
   - Extending UI to allow sound insertion: 3 hrs.
   - Integrating with the web services: ? hrs.
   - Validation of functionality and testing: 4 hrs.
   - Code-review, last refactorings and release: 4 hrs.

4. Adding image search in the application: 39 hrs. [40 hrs.]

   - Analyzing Google API/Interface to search for an image: 2 hrs.
   - Developing a program to fetch image results from Google: 10 hrs.
   - Developing UI for the user to request for image: 5 hrs.
   - Developing UI to show multiple results, fetched from Google: 10 hrs.
   - Reusing the reformatting algorithm from the adding image part: 5 hrs.
   - Validation of functionality and testing: 4 hrs.
   - Code-review, last refactorings and release: 4 hrs.

5. Loading from web pronunciation of words: 63 hrs. [40 hrs.]

    - Research on the services to be used: 5 hrs.
    - When concrete services are defined, research on the APIs to use: 10 hrs.
    - Developing UI for the user to request for sound: 5 hrs.
    - Developing a program to fetch sound results from services: 20 hrs.
    - Developing UI to show multiple results, fetched from services: 10 hrs.
    - Reusing the reformatting/compression algorithm from the adding sound part: 5 hrs.
    - Validation of functionality and testing: 4 hrs.
    - Code-review, last refactorings and release: 4 hrs.

6. Extending text editing with some rich text features: 58 hrs. [estimated to take "a lot of time", depending on features]

    - Planning, which basic richt text features are necessary (initially i, b, u, colors) : 1 hr.
    - Designing architecture, way to implement, to be compatible with other Anki clients: 2 hrs.
    - Implementation of basic rich text features: 40 hrs.
    - Validation of functionality and testing: 10 hrs.
    - Code-review, last refactorings and release: 5 hrs.

7. Integrating support for dictionaries/information sources: 99 hrs. [24 hrs. per source]

    - Analyzing existing application abilities, to determine best source (language, type of cards): 1 hr.
    - Research on existing applications, supporting similar functionality, and their implementation (GoldenDict): 3 hrs.
    - Research on the potential sources to include and their APIs (Wikipedia, Online dictioaries): 5 hrs.
    - Universal algorithm and UI to ask user for certain source, getting result back on the cards, using richt text features: 10 hrs.
    - Support for each source: 20 hrs. [24 hrs.] x number of sources (3 for a start) = 60 hrs.
    - Validation of functionality and testing: 5 hrs.
    - Code-review, last refactorings and release: 5 hrs.

2.3 Overall time estimation and reasons for deviations

Taking each ? to be 5 hours we end up with 329 hours approximately. This is an approximation from below, or optimistic approximation. To be on the safe side, not having more than 400 hours of work, we suggest, this must be enough for the project. Important to notice, is that the tasks above do not include documenting, which may be necessary, and not only for the course purposes, but also for the Anki project itself. Another not included in the tasks, but very important for calculating time issue is bug fixing. We may need to fix existing bug, before we proceed with new feature.

Potential danger in the project run, as we see now, is the dependency of the features. For example, loading pronunciation from the Web depends on a feature to have sound included in the card. Thus, having delays with initial features, it could be that the some of the features will not be implemented on time. As a justification for such delays, on the case they happen, we plan to provide technical report, explaining the difficulties on the way, causing the delay (for example, bigger than 5 hours time for ? items in the list).

2.4 Additional features

If the development goes well, and better than we expect, we have also many additional, "nice to have" features, which we could implement:

1. Better UI support for Tablet devices, and if possible-Google TV (Tablet device would be needed)

2. Support for finger drawing (complicated UI, no standard intents)

3. Accessibility for people with disabilities (research is needed first)

4. Fixing the list viewing (current well-known bug, not hard to implement)

5. Fixing "white squares bug" - some new fonts are needed to display all languages (hard task, because includes installing new fonts)

6. Support for videos (research for necessity is needed, similar to sound and pictures issue)

3 Suggested working style

References

Anki. Google code website, wiki with feature requests http://code.google.com/p/ankidroid/wiki/Index.

Anki. Homepage http://ankisrs.net/.