Testing the usability of the Android `ListView` component

A. Lautenbach

September 28, 2011

Abstract

Since the usability of an Application Programming Interface (API) is a major factor of its success, API design needs to be driven by usability concerns. The usability of an API can be evaluated in a similar way as the usability of other products. Usability testing with real users is the best method to evaluate usability.

The Android operating system for mobile phones has become popular among users and application developers. With the open Android API, the development of `ListView` components seems harder than the development of other GUI components. So in order to reveal usability defects in its interface, a usability test for the scenario of developing a `ListView` with icons was conducted with 5 test users.

One usability problem was identified, namely a mismatch of the level of abstraction provided by the interface and the level of abstraction expected by the users. The addition of an extra class to meet the users expectations or the improvement of the existing documentation is proposed as possible solutions.

1 Introduction

The field of software engineering has grown rapidly over the last decades, and efficient code reuse has become a major force to tackle the inherent complexity. In order to facilitate code reuse, many software components are modularised so that they can be used independently, employing the principle of information hiding as coined by David Parnas [19].

An application programming interface (API) provides an abstraction for application developers, exposing the functionality of specific modules or products. An application developer can use them in spite of being completely unaware of their internal structure or implementation.

One of the key realisations regarding API design is that it is a domain to which human factors and usability-engineering principles apply, suggesting that it should be a user-centric activity. This is a simple and yet often neglected fact, as Ken Arnold pointed out [1]. Market leaders such as Microsoft have recognised this, and have started to employ usability-engineering principles to guide their API-design processes [8].

The API of a successful product will be used often and in diverse contexts during its life time. This leads to some general requirements and guidelines which have been identified by industry experts such as Joshua Bloch [6] and Krzysztof Cwalina [9]. These guidelines are widely accepted by the developer community [4, 16] and are the subject of recent scientific research, an overview of which was made by M.F. Zibran [23].

In the last few years, the use of so-called "smartphones" for more than mobile telephony has increased greatly. Smartphone is the term used for a high-end mobile phone which is a hybrid of a personal digital assistant (PDA) and a mobile phone. Recent smartphones also feature multimedia capabilities such as music and video players, fast internet access and cameras. As it has often been speculated in the past, the switch from mobile telephones to mobile personal computers has begun. Mobile applications for all kinds of scenarios are becoming increasingly important. The mobile application market is still expected to grow in large steps and thus attracts much attention and many investors, since it holds the promise of high profit margins.

A multitude of operating systems for smartphones have been developed in recent years, such as Symbian, webOS, Blackberry OS, iOS, Android, Windows Phone 7, and others. Their manufacturers are competing for users and market shares as this usually translates to more revenue.

The availability of applications which simplify users' lives is often an important decision factor for or against a certain platform. A single platform provider could not possibly provide for all the needs of their users, so it is important for them to have open APIs to allow third-party application development. This also implies that the platform and the APIs should be attractive for mobile developers, so that many of them will start to write applications for the platform in question, thus increasing its market value.

There are mainly two different kinds of mobile developers. Some are hobby programmers who write applications just for fun, for their own and others amusement. The other group consists of professional developers writing commercial software. These two groups are not mutually exclusive, but there is usually a difference in how they approach the decision which platform to support with their applications.

The most important decision factor for professional developers is the potential user base, i.e., the market share of the platform and its expected market share in the future. In addition, they prefer to develop for multiple popular platforms to further increase their potential user base. Another important goal for them is to keep the development time short to maximise profits, therefore ease of development is also of importance. Ease of development is usually achieved through good development tools and clear, complete and well-documented APIs.

For hobby developers on the other hand, personal platform preference and ease of development are usually of paramount importance. They usually only develop their applications for a single platform.

Therefore, clear, complete and easy to use APIs can have a big impact on the success of a mobile platform as a whole, which is actually also true for products and technologies in general [5, 6, 15, 16, 18, 21].

As part of the team which designed the GravityZoo API for the GravityZoo platform, I got professionally interested in API design and the process of designing APIs with a high usability. For obvious reasons, usability is the prime design directive. While working on the GravityZoo client for the Android platform, I also got well acquainted with different Android APIs, including parts of the GUI APIs. One of the components I developed was a wrapper for the Android `ListView` which could take both text and corresponding icons as input. A `ListView` is a list of scrollable and clickable components. A number of different classes and components are needed to construct a fully functional `ListView` component. The `ListView` component turned out to be less intuitive than most of the other GUI parts. As a result, I started to investigate the cause of this difficulty.

Therefore, the problem statement for this bachelor thesis reads as follows: *What usability issues exist in the interface of the* `ListView` *component of the Android API, and how could they be alleviated?*

Two research questions need to be answered in order to be able to address the problem statement. First, how can you define a user-friendly API, and secondly, how can the usability of an API be measured.

The general outline of the paper is as follows. First a general background for API usability issues will be established in section 2. To place the `ListView` component in its context, a short introduction to the Android operating system is subsequently given in section 3. Then the setup of the usability tests will be described (section 4), which is followed by a discussion of the background of the test users (section 5). Subsequently the results of the tests will be presented (section 6). Finally, some conclusions will be drawn and recommendations for further research will be given in section 7.

2 API Usability

In this section the research questions what a user-friendly API is and how you can measure API usability will be addressed.

2.1 Characteristics of User-Friendly APIs

As was argued before, API design is an activity to which usability-engineering principles apply [1, 8]. This implies that the usual definitions and techniques from usability-engineering can be used as reference points to develop the concepts of user-friendliness and usability measurements tailored specifically to API design. It also means, that the whole design process needs to be user-centric, in order to achieve the highest possible usability.

Usability expert Jakob Nielsen mentions the following attributes as constituting the field of usability [18]:

- Learnability
- Efficiency
- Memorability
- Errors
- Satisfaction

The point "Errors" refers to the user making as little errors on the way to the desired outcome as possible, whereas "Satisfaction" is about user satisfaction, i.e., the user should like to use the product in question. For a more in-depth discussion of all these points, see chapter 2.2 of Nielsen's book *Usability Engineering* [18].

Additionally, Java and API design expert Joshua Bloch, formerly Distinguished Engineer at Sun Microsystems Inc., nowadays Chief Java Architect at Google Inc., recognizes the following elements as constituting a "good" API [6, 7]:

- Easy to learn
- Easy to use, even without documentation
- Hard to misuse
- Easy to read and maintain code that uses it
- Sufficiently powerful to satisfy requirements
- Easy to evolve
- Appropriate to audience

It is quite obvious that there are direct correlations between these two lists, the latter seemingly a refinement of the former.

"Know your user" is one of the most important guidelines in usability-engineering [18]. It should be obvious that APIs are used by programmers, so the user base is already narrowly defined, which makes it possible to make a couple of assumptions. Yet programmers are rather diverse and have different mentalities and workflows. Even though they all may write software for a living, an embedded-systems programmer usually has different needs and expectations than a programmer of business software or middleware, just as a senior programmer usually has a different work-flow or problem-solving approach than a beginning programmer. Consequently, in order to be most effective the context of an API and its expected user base should also have a high impact on its design.

Many API-design guidelines [4, 6, 9] and other publications related to API design [5, 16] exist, but little research has been done to falsify them. Recent research has picked up on this, trying to scrutinise claims, idioms and patterns commonly accepted as correct. For example, the impact of the design patterns Factory Method and Abstract Factory on API usability has been examined [10], as well as the implications of parameters in object constructors [20] and where to place a method in an object-oriented API [21].

It should be noted that API stability naturally follows from high usability. If an API is easy to use and there is good error prevention, i.e., it is hard to misuse, then the need for change is minimal. Additionally, a good API is expected to evolve so that changes integrate easily without breaking backward compatibility.

2.2 Measuring API Usability

According to Nielsen [18], one of the best ways to measure usability is to conduct usability testing. Usability testing is done by testing a product through direct user interaction. During a usability test a group of test users will be confronted with one or more concrete usage scenarios for the product in question. The scenarios should be designed in such a way that they capture the essence as well as the full scope of the tasks which should be accomplished with the product. This implies that usability testing is a relative measure with respect to the testers participating in the study and the tasks the testers have to accomplish.

Usability testing can also be applied to measure API usability. According to Jeffrey Stylos, Steven Clarke and Brad Myers, qualitative studies using the "thinking aloud" method [17, 18] have yielded good results in practice [22].

In order to analyse the results from API usability studies, people from Microsoft's usability labs have successfully used an adaptation of the cognitive-dimensions framework [8]. The cognitive-dimensions framework was originally proposed by Green and Petre to evaluate structures in terms of their cognitively relevant aspects [13, 14]. In the design phase it can be used as a heuristic to guide the design process, and when a usability study has been conducted it can help to interpret the results.

Personas [14] are a theoretical instrument to define a set of typical characteristics of different personality types. Following the example of the Visual Studio usability group, we will distinguish the personas of opportunistic, pragmatic and systematic programmers, as described in [8, 14, 20].

It can be assumed that developers using the Android APIs have a background in developing mobile or web applications. These two fields are mainly driven by rapid application development, which implies that the most common types of developers will be pragmatic and opportunistic programmers.

3 The Android Operating System

In order to understand the context in which the `ListView` component is used, a short overview of the Android platform, its design and use of certain GUI APIs will be given here. Next, the Android `ListView` component will be described.

3.1 The Android Platform

The Android operating system developed by Google Inc. has managed to gain a considerable share of the smartphone market in just a few years. This is partly attributed to the large amount of applications available in Google's software store called Android Market, which is the result of an open platform that is popular among application developers due to easy-to-use APIs, development tools and the use of the popular programming language Java. According to Google, there are more than 200,000 applications available in the Android Market [2].

Operating systems for smartphones have some rather unique requirements, constraints and properties in general. In comparison to desktop computers or laptops, the computing power and memory space of smartphones are quite limited. Moreover, depending on the specific product their hardware equipment can differ greatly. Display size and quality often vary, and some phones have a camera, GPS and other optional components, whereas others have none or only a subset of them. The trade-off between phone size, weight, battery run-time and computing power needs to be considered, too. Frequent location and network changes also pose interesting challenges. Therefore, operating systems for smartphones

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">

  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />

</LinearLayout>
```

Listing 1: `res/layout/main.xml`

have to be both flexible and efficient.

Android knows four different types of application components, namely Activities, Services, Content Providers and Broadcast Receivers. For the usability test discussed in this paper only activities are relevant, so we will ignore the rest.

An activity is the basic building block for user-interfaces on Android, because one activity corresponds to a single screen. So an application with a complex user-interface usually consists of multiple activities which are stacked upon each other, which is then called the "Back Stack".

An activity has hooks for different states of its lifecycle, such as its creation and destruction. The only hook of interest in the tasks of this usability test is the `onCreate` hook, which is called after the activity has been fully constructed. It is often the main starting point for a user-interface-based application.

Another important device for GUI programming on Android is the use of XML files to construct user-interfaces. This can best be demonstrated with a simple example. Eclipse is the standard IDE used for Android development. When a new Android project is created in Eclipse, it will contain the template of a simple Hello-World application. The application uses a `TextView` in a `LinearLayout`, which is comparable to a `JLabel` in a `FlowLayout` of the Java Swing libraries. However, this GUI structure is not defined in the Java code, but in an XML file. The Android plug-in for Eclipse automatically parses and compiles the XML files in the project resource folder `res/layout` during the build. The result is a static resource class called R, which provides access to different resources, such as references to GUI components constructed from an XML layout file.

Listing 1 shows the content of the XML file from the standard Hello-World template. Note that the `TextView` is embedded in the `LinearLayout`. Its text refers to another resource, namely another XML file

```
public class TestActivity extends Activity
{
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }
}
```

Listing 2: excerpt from `TestActivity.java`

(`res/values/strings.xml`) which contains a name-value pair for the key "hello" with the content "Hello World, TestActivity!".

Listing 2 shows the Java code of the activity which refers to the XML layout file in order to construct its GUI. `R.layout.main` is an integer reference to the layout file `main.xml`, and `setContentView` sets the main view of the activity. The top-most component in `main.xml` is the `LinearLayout`, so effectively the `LinearLayout` containing the `TextView` is set as the view of the activity. The text "Hello World, TestActivity!" will be shown when the application is started.

In combination this accomplishes the same as the code shown in listing 3.

For an in-depth discussion of the different features of the Android platform, see the official Android developer documentation [12].

When browsing the Android APIs, it quickly becomes apparent that many of the recent findings for good API design have already been applied, e.g., the create-set-call pattern as described in [20].

3.2 The Android `ListView` component and the adapter pattern

The main part of the `ListView` component is the class `android.widget.ListView` itself. It is responsible for displaying a list of scrollable and clickable items, but the data for the items comes from an `android.widget.ListAdapter`.

`ListView` is a subclass of `AdapterView`, which is an abstract view component. An `AdapterView` receives its data from an `Adapter`, which in turn is an interface to be implemented by a concrete adapter class. This is the adapter pattern at work, as described by Gamma et al. [11]. It decouples the view from the model, because only the adapter needs to know about both the model and the view in order for the view to be able to display the data. The model and the view can remain unchanged. This is a common pattern which is also extensively used for Eclipse plug-in development and the Eclipse Rich Client Platform. A good explanation of the basic principle can

```
public class TestActivity extends Activity
{
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    TextView tv = new TextView(this);
    tv.setText(R.string.hello);

    LinearLayout ll = new LinearLayout(this);
    ll.addView(tv);

    setContentView(ll);
  }
}
```

Listing 3: Hello-World example without XML GUI

be found in [3].

For the `ListView`, a number of predefined adapters exist for different data sources, for example `ArrayAdapter` for a simple list of items, or `CursorAdapter` which wraps a cursor object holding the results of a database query.

There is also a special activity which simplifies the construction of an activity using a `ListView`, `android.app.ListActivity`.

For the exact interfaces exposed by the aforementioned classes, see the official API reference [12].

4 Usability Testing

The following sections will introduce the test setup, the test environment and the test tasks, in that order.

4.1 Test Setup

The usability test was set up as a qualitative study using the "thinking-aloud" method [17, 18, 22]. Its goal is to evaluate the usability of the `ListView` component. According to Nielsen a number of five test users provides a reasonable trade-off between finding usability problems and the time needed to conduct and analyse the experiments. See chapter 6 "Usability Testing" of Nielsen's book for more details [18]. The five test users which were chosen are personal acquaintances of the experimenter and have no prior experience with mobile application development. They will be referred to by the aliases User1 to User5.

The tests were all done in different locations, either at the test user's home or at his work place, depending on his preference. However, the tests were all done with the same laptop, the same mobile phone and the same software setup, which will be discussed in more detail later.

The tests were designed so that novice users should be able to complete all tasks within the time limit of one hour, in order to keep the produced data manageable and the test users interested. If a user did not complete the tasks within one hour, the test was stopped.

The most basic measure taken during the tests was the time a user needed to complete each task. A task was viewed as completed when an application which satisfied the requirements was successfully executed on the mobile phone.

Screen casts were recorded during all tests, so that video material was available which could be reviewed and analysed after the test. Audio was not recorded, but the experimenter took notes of his observations during the tests, especially those that would not show on the video such as questions or reasoning for specific actions.

The testers were allowed to use any resource available to them, which explicitly included search engines and online resources. This was decided on the basis that nowadays pretty much all programmers have constant broadband internet access, and prohibiting access to it would lead to artificial test conditions which are not based on real usage scenarios.

Questions from the test users were encouraged during the tests, but they also were informed that only general questions relating to the `ListView` component could be answered by the experimenter. However, the experimenter was allowed to give tips and help on issues that were not directly related to the interface of the `ListView` component, such as the keyboard short-cut in Eclipse to organise imports automatically.

Before the tests, all users were given a questionnaire with questions about their general programming background and about their knowledge of the specific task domain. They also received a few follow-up questions after the tests, such as to their former experience with the adapter pattern.

It was also explicitly explained to the testers that the usability of the `ListView` component is to be evaluated, not the performance of the tester. Moreover, they were made aware of the screen recordings and the general test procedure.

After the test, all users had the opportunity to give feedback via a simple feedback form pertaining to the tasks and the test procedure.

The tasks for a usability test should be designed in such a way that they are as representative of the actual use as possible. For programming in general this includes the tasks of code reading, understanding, adaptation and writing. For Android `ListViews` in particular however, there is no data available which would reveal typical usage scenarios. The time limit of one hour also excluded a number of more complex scenarios which would require more foreknowledge. Consider for instance a `ListView` using the `CursorAdapter`, which next to the adapter pattern also uses the cursor pattern, usually in correlation with a database.

The scenario of implementing a `ListView` with a custom icon and text per list item was chosen because it seemed to have the appropriate level of difficulty.

4.2 Test Environment

The test environment was set up as follows. The computer on which the experiments were conducted was a Lenovo ThinkPad T500, with an Intel Core 2 Duo P8700 processor @2.53 GHz, 4 GB of RAM and a 15.4" widescreen with a resolution of 1680x1050, using Windows 7 Professional as operating system.

The smartphone used as the test device was an HTC Desire running Android 2.2. The programs could also have been tested in the Android emulator which is part of the Android SDK, but using a real device is faster and usually perceived as more fun for the tester. However, for the analysis of the usability test this also had the disadvantage that the user interface of the device was not captured with the screen cast.

The screen casts were recorded with CamStudio version 2.6b.

Google Chrome version 12 was opened with 3 pre-set tabs before the tests began and was explicitly brought to the test user's attention. The tabs had windows open to the Android API reference, to the reference for the `ListView` class, and to Google's official `ListView` tutorial.

The Android SDK and Eclipse with the Android plug-in were also pre-installed, including the necessary HTC Desire device drivers. Every tester used a new workspace with the same settings, and every task of a test was developed in its own Android project.

4.3 Test Tasks

In the following the three individual tasks will be discussed in detail.

Task 1: Hello-World application

The first task instructed the user to write a Hello-World application with the use of a `TextView`. As discussed in section 3, a new Android project in Eclipse already contains a Hello-World template. Therefore, the goal of the task was to write the same application without defining the GUI in an XML file. A possible solution was already shown in listing 3.

Apart from being a warm-up task, it required the test user to go through the whole build process at least once and let him run a simple program on the test device. This gave the user an early success experience and familiarised him with the basic development process.

There is an official Hello-World tutorial which describes a solution to this task, so it could theoretically be solved by just copying and pasting the code from the tutorial.

Task 2: `ListView` with text

The second task was designed to familiarise the test user with the use of the `ListView` component. The task was to write a simple `ListView` with eight given European city names as items. The simplest solution for this task is described in the official `ListView` tutorial and involves an `ArrayAdapter`. So once again copy and paste would have been enough to solve the task.

`ArrayAdapters` need two pieces of data. First, they need a reference to the `TextView` which should be used to display the items. The `TextView` has to be defined in an XML file. Secondly, the `ArrayAdapter` needs an array or list of objects which represents the data. On each data object, `toString()` is called to retrieve the text to set on the `TextView`.

The idea behind this task was to give the test user another early success story, while also introducing him to the basic concepts used in the `ListView`. That is, the use of the adapter to combine the view with the data, and the use of references of view components defined in XML to construct more complex GUI components.

Task 3: `ListView` with text and icons

Finally, the last task extends the former task by requiring that a different icon be placed next to each city name given in the previous task. The icons were provided. This is the task which is expected to require most of the time, as the solution can not be readily found in the official documentation, and there is no trivial solution.

This task requires all the basic tasks which programmers most often have to master when dealing with an API: to read and understand existing code, to adapt the existing code and to write new code.

Since this is by far the most complex of the three tasks, it is also expected that it will help to identify the most usability issues. The other tasks have been rather easy so that they are expected to only reveal usability issues of little significance.

5 Background of the Test Users

In order to be able to understand the test results, the background of the test users has to be taken into account.

5.1 General educational and programming background

All of the users have a similar academic background, since they did the same bachelor study at Maastricht University, and three of them also did the same master. Three are currently PhD students, one is a master student who is about to graduate and one is a bachelor student who is also in the later stages of the study.

In a short depiction of how the persona of a systematic, pragmatic and opportunistic programmer would approach a programming task in general, two would categorise themselves as being opportunistic, and the other three as being pragmatic.

The number of years of programming experience in general ranged from 6 to 12 years, with a mean of 9.2 years. On average they have programmed in 5.8 different programming languages, with Java (5 of 5), Matlab (4 of 5) and C++ (3 of 5) being the most common.

Their mean number of years of experience with Java in particular is 7, and four of the five testers have used Java as their main programming language during their last project. For three of them, their last non-trivial programming experience with Java was less than a week ago, and for the other two it was between 3 and 6 months ago.

Two test users indicated that they have programmed for a commercial project at least once, and three declared that they also did hobby projects next to their university projects.

Some basic software design patterns are known to all, and all but one have used at least some of them in production code already. The theory of the adapter pattern is known to four, but only two have applied it in production code so far.

5.2 Knowledge of the task domain

All test users had used the Eclipse IDE before, so it needed no introduction. Although two of the testers had not used Eclipse in more than two years, the others had used it recently.

GUI programming for Android and other mobile platforms, or with the use of XML layout files, was new to all the participants. Therefore, they received a very short introduction to some basic Android development concepts from the experimenter before the tests commenced. This included a brief explanation of Activities, a demonstration of what a `ListView` component looks like and how a new Android project can be created in Eclipse. But most importantly, a quick explanation of the use of XML layout files in Android and the use of the local resource class `R` was given.

During the tests the experimenter provided help and tips when they were unrelated to the task at hand. Moreover, the experimenter also provided some more extensive help when it came to the general understanding of the XML layout files, and how they can be used. This certainly influenced the test results, but knowledge of the use of XML layout files can usually be assumed before a `ListView` component is written by an Android developer. The reasoning is that the use of XML layout files is a very basic technique on the Android platform, which is integrated in all official tutorials from the beginning. But `ListViews` are slightly advanced components which are usually not used by total beginners. Therefore, the help provided for XML layout files by the experimenter was considered to bridge that gap.

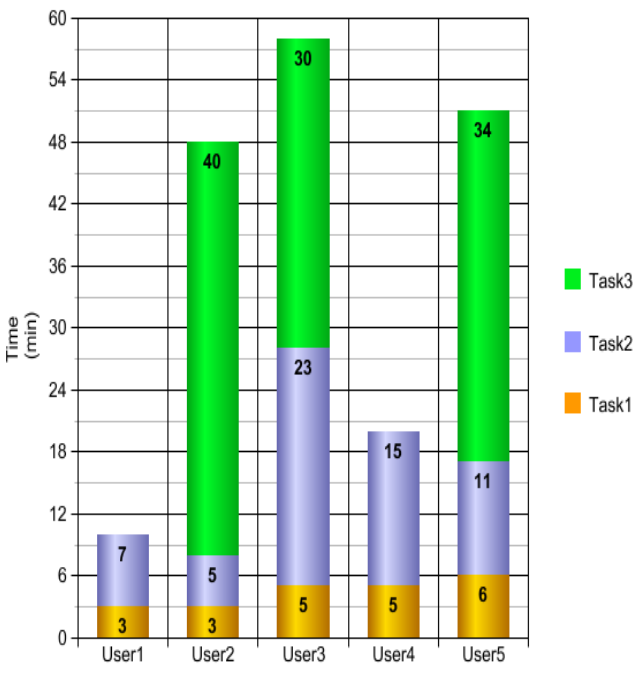

Figure 1: Completion Times

6 Usability Test Results

In the following, the test results will be discussed per task.

6.1 Task 1 Results: Hello-World application

All the participants finished task 1 in 3 to 6 minutes, without any problems. An overview of the times per task per user can be found in figure 1.

Three testers explored the reference of the `TextView` class to figure out the correct methods to call and wrote the Java code themselves. The remaining two simply copied the code from the official Hello-World tutorial. There were no surprises here.

6.2 Task 2 Results: `ListView` with text

As explained in the task description, task 2 required the use of an XML file containing a `TextView`, which could be referenced in the `ArrayAdapter`. All testers struggled a little bit with understanding the concept and how it can be used for user-interface construction. However, with the help of the official ListView tutorial and occasional explanations from the experimenter they all figured it out in the end.

The time to completion differed greatly for this task. The mean of the completion time was 12.2 minutes, but the fastest tester finished after 5 minutes whereas the slowest needed 23 minutes.

However, User3 who needed the most time slightly misunderstood the task at first. He thought he was not allowed to use XML layout files, so he started to look for more complex solutions than was necessary. So this was in part a failing of the experimenter. Moreover, when User3 had already completed the task, a defect of the automatic build feature in Eclipse prevented him from compiling the solution, which prompted him to assume that there was something wrong with his code, which was not the case. It was resolved by a hint from the experimenter to try to clean the project, but the user had already lost another 5 minutes to this issue.

User4 started out with examining the interface of the ListView API and looked for an append method or something similar to add elements directly to the ListView object. When that was unsuccessful he also turned to the ListView tutorial. This reveals that User4 expected the ListView to contain its own data elements.

User1, User2 and User5 started directly with the ListView tutorial and more or less just copied the solution, so there was not much to be learned about the use of the interface.

6.3 Task 3 Results: ListView with text and icons

Both User1 and User4 did not manage to finish this task in the given time limit. But User4 had a clear path to the solution when time ran out, he probably would have needed only a few more minutes to finish. User1 seemed rather lost, though.

Only counting the people who completed the task, the mean of the completion time was 34.3 minutes, with a minimum of 30 and a maximum of 40 minutes.

After figuring out that the ListView tutorial does not cover this scenario, the first step of all participants was to look for a simple constructor of the ArrayAdapter which could take both text and images as input data. Failing to find this, they all turned to Google to look for answers, and many different solutions were found that way.

One of the easiest and most popular ways to implement a ListView with custom icons is to subclass ArrayAdapter and override its getView method. Unfortunately, the documentation for the getView method is non-existent in the API reference of the ArrayAdapter. One has to go to the reference of Adapter to get a proper description of that method. The getView method receives an index of the data for which a view is requested, and returns the view to display for that data item. So to solve the task, the view which should be returned by the getView() call should consist of a LinearLayout containing an ImageView and a TextView. Then the text of the TextView, and the icon depicted with the ImageView can be dynamically adapted during the getView call, based on the position of the item.

An intermediate step taken by almost all participants was to first develop a ListView which used a single icon for all items. This only requires a minimal adjustment of the solution of task 2. The XML file from solution 2 has to be changed so that it contains a LinearLayout holding a TextView and an ImageView with a hard-coded reference to the icon which should be used. Then only a different constructor of ArrayAdapter has to be used, namely one which takes a complex view containing a TextView, and the ID of that TextView. Then the ArrayAdapter will return the complex view as the item, and will only adapt the content of the TextView, identified by the given ID. Every participant got at least this far. But User1 kept looking for a simpler solution than writing a custom adapter, finding it hard to believe that such a seemingly simple task should have to be so complex.

Only User2 seemed to understand the relationship between the ListView and the different ListAdapters immediately. Once you realise that the adapter pattern is used there, it should be obvious that either an adapter must exist which is sufficient for the task, or the solution must be reached by implementing an adapter yourself. User2 did state that he knew this particular usage pattern of the adapter from the Eclipse Rich Client Platform (RCP) API. But he also wasted quite some time looking for a simpler solution, hoping that he could avoid to write a custom adapter.

None of the other participants seemed to make the connection to the adapter pattern, at least no one even mentioned the adapter as a pattern in this context. On the other hand, this is completely ignored in the Android documentation, too. Unless you have previously been exposed to this specific usage scenario of the adapter pattern, the connection is not obvious.

By now it should be clear that there was a definite mismatch of what the test users expected from the interface and what the ListView component offered.

What seems a little bit surprising is that almost no one looked further than the ArrayAdapter; other adapters such as SimpleAdapter were not even considered.

6.4 Test Feedback

The feedback form which was presented to all participants after the test yielded a few notable results.

The question whether the tester liked to use the API resulted in three positive, one neutral and one negative answers.

The API documentation, however, was declared insufficient by all participants, especially with respect to task 3. Only User2 complained about a lack of documentation of the XML features instead of a lack of documentation for `ListViews` with icons, who incidentally is also the only user who understood the underlying pattern and has an extensive commercial background.

One participant also stated that task 3 was too difficult because it required too much background knowledge.

7 Conclusions and Future Research

The conducted usability test only revealed one major usability issue, namely the mismatch in the level of abstraction exposed by the `ListView` component, and the level of abstraction users expect for the given scenario. The test users kept looking for an option to add icons to the `ArrayAdapter`. Displaying custom icons next to a list of items is a common scenario which they expected to be directly supported by the API, which is not the case.

The problem was that, even though the eventual solution is quite simple, it was not perceived as such by the people who did not recognize the underlying pattern.

To alleviate this problem, one could extend the official `ListView` tutorial with material on how to achieve the above-mentioned effect, and to improve the documentation of the `ListView` and `ArrayAdapter` classes. This is what the test users themselves suggested and would be the least invasive.

Another possible solution would be to add a subclass of the `ArrayAdapter` to the official API, for the special case of a `ListView` with icons and text, just as the `ArrayAdapter` is a special adapter for a `ListView` where only a `TextView` needs to change. The API designers probably considered this API bloat, because such a class can easily be written by the developer himself.

It is conspicuous that the testers characterised as being opportunistic both failed to finish within the time limit. This leads to the hypotheses that the above mentioned usability issue is worse for opportunistic programmers than for pragmatic programmers.

The validity of the performed tests is somewhat limited, because none of the test users are actual Android developers. It is further limited by the fact that they all have a highly similar academic background.

In the future, it would be interesting to repeat this usability test with two test groups, one using a solution with a custom `ArrayAdapter` subclass, and one with improved documentation and extended tutorials. This could provide insight into the question which of the two proposed solutions would be better suited for alleviating the discussed usability problem.

To determine whether the new proposed class should indeed be added to the API, it might be interesting to conduct a study on how many open-source projects currently use a `ListView` with icons, and what their perceived level of difficulty was for implementing such a component.

It would also make sense to repeat the usability test with a more heterogeneous test group to validate the findings. In addition, the tasks could be adapted to include pure code-reading tasks, and tasks which test for "debugability".

Acknowledgements

I would like to thank my supervisor Jos Uiterwijk from Maastricht University for his constructive feedback, ideas and patience.

Moreover, I would like to thank my former colleagues from GravityZoo, who also supported me while writing this thesis, especially in the first few stages while I was still employed at GravityZoo.

Finally, I owe a debt of gratitude to my family and friends for their reviews and general advice on writing a paper.

References

[1] Arnold, K. (2005). Programmers are people, too. *ACM Queue*, Vol. 3, No. 5, pp. 54–59.

[2] Barra, Hugo (2011). Android statistics. `http://googleblog.blogspot.com/2011/05/android-momentum-mobile-and-more-at.html`.

[3] Beaton, W. (2008). Adapters. `http://www.eclipse.org/ articles/article.php?file=Article-Adapters/index.html`.

[4] Blanchette, J. (2008). The little manual of API design. `http://chaos.troll.no/~shausman/api-design/api-design.pdf`.

[5] Bloch, J. (2001). *Effective Java. Programming Language Guide.* The Java Series. Addison-Wesley, Upper Saddle River, NJ, USA.

[6] Bloch, J. (2006). How to design a good API and why it matters. *OOPSLA '06: Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*, pp. 506–507, ACM, New York, NY, USA.

[7] Bloch, J. (2007). Google Tech Talks: How to design a good API and why it matters. `http://www.youtube.com/watch?v=aAb7hSCtvGw`.

[8] Clarke, S. (2004). Measuring API usability. *Dr. Dobb's Journal Special Windows/.NET Supplement*, Vol. 28, No. 05, pp. S6–S9.

[9] Cwalina, K. and Abrams, B. (2008). *Framework Design Guidelines*. Microsoft .NET Development Series. Addison-Wesley, Upper Saddle River, NJ, USA.

[10] Ellis, B., Stylos, J., and Myers, B.A. (2007). The factory pattern in API design: A usability evaluation. *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pp. 302–312, IEEE Computer Society, Washington, DC, USA.

[11] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns. Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, Indianapolis, IN, USA.

[12] Google Inc. (2011). Android API reference. `http://developer.android.com/reference/`.

[13] Green, T. R. G. and Petre, M. (1996). Usability analysis of visual programming environments: A "cognitive dimensions" framework. *Journal of Visual Languages and Computing*, Vol. 7, No. 2, pp. 131–174.

[14] Green, T. R. G., Blandford, A., Church, L., Roast, C.R., and Clarke, S. (2006). Cognitive dimensions: Achievements, new directions, and open questions. *Journal of Visual Languages and Computing*, Vol. 17, No. 4, pp. 328–365.

[15] Henning, M. (2006). The rise and fall of CORBA. *ACM Queue*, Vol. 4, No. 5, pp. 28–34.

[16] Henning, M. (2007). API design matters. *ACM Queue*, Vol. 5, No. 4, pp. 24–36.

[17] Lewis, C. and Rieman, J. (1993). Task-centered user interface design. `http://hcibib.org/tcuid/`.

[18] Nielsen, J. (1993). *Usability Engineering*. Morgan Kaufmann, San Francisco, CA, USA.

[19] Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM*, Vol. 15, No. 12, pp. 1053–1058.

[20] Stylos, J. and Clarke, S. (2007). Usability implications of requiring parameters in objects constructors. *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pp. 529–539, IEEE Computer Society, Washington, DC, USA.

[21] Stylos, J. and Myers, B.A. (2008). The implications of method placement on API learnability. *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 105–112, ACM, New York, NY, USA.

[22] Stylos, J., Clarke, S., and Myers, B.A. (2006). Comparing API design choices with usability studies: A case study and future directions. *PPIG 18: Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group*, pp. 131–139.

[23] Zibran, M.F. (2008). What makes APIs difficult to use? *International Journal of Computer Science and Network Security*, Vol. 8, No. 4, pp. 255–261.