University of Glasgow | School of Computing Science

Level 3 Project Case Study Dissertation

An Example Project

Jordyn Anne Brown
Nathan Gordon Kirkpatrick
Muhammad Raza Ali
Scott Brown
Josep Perna Montane
Ruairi Gielty

6 April 2020

Abstract

The abstract goes here

Education Use Consent

We hereby give our permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format.

1 Introduction

Software engineering

This paper presents a case study of...

The rest of the case study is structured as follows. Section **??** presents the background of the case study discussed, describing the customer and project context, aims and objectives and project state at the time of writing. Sections **??** through Section 11 discuss issues that arose during the project...

2 Case Study Background

Include details of

- The customer organisation and background.
- The rationale and initial objectives for the project.
- The final software was delivered for the customer.

3 QR Code Implementation

On the second customer meeting, Obashi told us that we should scatter the cubes in a wider area. The research done and different approaches taken for solving this problem are explained in the Cube Scattering section.

After doing research and trying the different approaches, we decided to use a QR code that would provide our application with the size of the room so that cubes could be scattered accordingly. To do this, when starting the application we place the QR code in the centre of the room and scan it before entering the AR Activity.

This left us with the need of implementing a QR reader in our application. Being this something that is implemented in many apps we decided to use ZXing[13], an existing QR scanning library. Implementing it was pretty straightforward thanks to all of the documentation provided in the repository. The only blocker that we found was that we had find a way of passing the measures retrieved from the QR scanner to the AR view. We used the extra field of the intent to do this. The extra field, contains a dictionary in which we could store the measures and then they could be retrieved in the AR activity and used to scatter the cubes.

4 Cube Scattering

Initially when it came to scattering the cubes, a simple algorithm to calculate a random position on the plane detected by ARCore was used. The functions `getExtentX()` and `getExtentY()` in the `getRandomPlacement` method in the `ArUtils.java` file return the estimated lengths for each local axis centred on the plane, therefore this must be doubled for both `+` and `-`. A translation of the center of the plane is then used to create an anchor node which is then assigned the cube to later on. However with this basic implementation two serious blockers emerged.

The first major blocker was that cubes were sometimes being placed inside each other, a simple fix devised for this was to keep a list of pairs of all cubes and their anchor nodes positions and if a new cube was needing to be placed then iterate over this list and check whether or not the new position was within a minimum distance of any other cubes and either generate a new position and check again or place it.

The second major blocker encountered was that not all the cubes that were desired to be placed were being placed, the first assumption was that this was due to the distance trying to be maintained between all the cubes but later discovered this was due to the plane not being large enough to fit all the cubes in. For the plane detected by ARCore to grow, the user needed to pan the camera and ARCore needed to detect that the plane can be expanded - this can therefore create a situation were the user is searching for the last cubes which have not been placed due to the plane not being large enough but the user is panning in such a way that ARCore is not recognising the plane and the game being stuck in a limbo state. This issue was overcame with the implementation of the QR Scanner (and later manual integer input) which negates the need for checking the plane extents and simply use a `distanceToPlace` variable which receives an integer in the form of the radius of the room from the scanner or manual input to determine the dimensions of the room to scatter accordingly. This implementation also allows for all the cubes to scatter instantly as there is no need to wait for ARCore to grow the size of the plane.

When it came to assigning elements from the diagram to the cubes it was decided that there was to be a list of the elements from the diagram, `listToStack` which would then be used to create a stack of all the elements, `Collections.shuffle` is then used to randomise the order of the stack and then whenever a cube was needing to be placed and the position was okay an element could be popped from the stack and assigned, with the number of cubes needing to be placed coming from the difficulty selection on the main screen.

A design decision made further into development after the 4^{th} customer day was instead of having the user collect cubes with the diagram element text being displayed directly on them, instead there would be an `infocard` which would display the text and feature a button for the user to tap on for them to collect the element and add it to their inventory. This was done to increase the game aspect of the application and therefore it was decided that these cards would only be displayed once the user had found the cube and tapped on it and the element could only be collected if the user was within a certain distance of the cube - this distance is also determined by

the difficulty selection on the main screen.

When it came to the design of the cubes, it was kept simplistic but also designed in such a way as to challenge the user when it came to finding them. Therefore a random RGB value as well as a random transparency and size within a set range is assigned to each cube - this can be found within the `createCube()` method in the `ArActivity.java` file.

5 Inventory

It was decided that the renderable objects (the cubes) should be added to an inventory once collected by the user in the environment. This would enable the user to visually keep track of the number of objects gathered, and also help to monitor the number of cubes remaining in the environment.

The first design displayed the inventory as a labelled icon which could be selected on touch by the user to display an overlay. It was considered that this may be the best option as users would be familiar with the GUI of most apps and games. Thus, the GUI would follow convention and would not be too difficult for users to quickly understand its functionality.

However, during implementation a number of resulting issues appeared. First, the users would be a range of ages with a variety of experience. As the app was described to be used on phones or tablets, screen size also became an important factor. In the original design, the overlay would need to span half the size of the screen, while the other half would contain the overlay for the diagram. On implementation this appeared cluttered. Visually, the inventory became difficult to look at, and thus unappealing to users to complete the task. It quickly became clear that other options would need to be explored. During the first customer meeting, the client emphasised a desire to make the app as interactive as possible. Through all features of the app the aim was to fulfil this requirement, and the inventory was no exception. Thus, during implementation this request also needed to be considered.

With these requirements in mind, exploration began to find further options. Android Studio's horizontal RecyclerView widget was both visually stimulating and consumed little screen space. RecyclerView operates like a list. It provides the layout for the list and each list item in the xml file, which implements the CardView widget, placing each text object in a separate box to increase gamification of the app. To display the list on screen, it must be connected to a layout manager ('LinearLayoutManager') which is responsible for laying out each item in the view. Setting it to horizontal ensures that the list items are displayed as a row. An adapter must then be called to populate the views. In this case, the 'RecyclerViewHorizontalAdapter' is used. When an item is collected, its name is added to the list 'chosenCubes'. For each item in the list, a view holder is created. The adapter binds list items to view holders and is responsible for loading each item when the user scrolls through the list. It saves the view holders so that the user can scroll easily in both directions. With this set

up, each time the user collects a new cube, the name is added to the list, a new view is created for that item and displayed on the screen. The final design can be viewed in the presentation.

During implementation, it was found that Android Studio would not allow the full range of java capabilities with each data structure. The list of object names, now called listToStack, was originally implemented as an ArrayList. There was also a separate function to choose a random number within the range of objects. This then read a unique string from the list, assigned it as a name to the cube, and then deleted it from the list. This name would then appear in the inventory when the cube was collected. That is, until it was discovered that elements could not be deleted from ArrayLists in Android Studio. To solve this, the object names are initially added to a list. This list is then converted to a stack, shuffled, and the first element popped each time a new cube is created. This was the only technical problem encountered during implementation.

The resulting layout solved many of the aforementioned issues. The inventory adjusts easily to all screen sizes and orientations. Displaying collected items automatically at the bottom of the screen allows users of all experience to clearly identify what to do. Augmented reality is still a fairly new technology to most, so having a mix of 3D and 2D items prevents the app from appearing daunting to new users. A counter was added at the top of the screen to help users understand how many cubes were left to find, as the horizontal scroller now meant that this was more difficult just by glancing at the inventory. To establish consistency, list items display in the same Obashi orange implemented on the infocards. The list objects highlight and darken when selected or deselected by the user, which was another feature implemented to improve interaction and develop the gaming experience specified by the company.

Implementation of the final inventory design was fairly straightforward. However, agreeing on the correct design was surprisingly the most time-consuming issue. Multiple attempts were made at implementing a number of the designs, which on reflection, wasted a lot of time which could have otherwise been spent implementing the diagram overlay. In total, the inventory spanned two customer meetings, which was anticipated by the group and planned for initially. However, following implementation, it was clear that time could have been reduced on this feature. It became increasingly regrettable as other issues required more work than originally anticipated. A significant portion of time was spent trying to implement multiple designs. This was a result of learning how to use Android Studio and ARCore while building the app and so, much delay was due to research. Other minor issues revolved around having to meander the software which was still in the stages of construction. It truly did require continual maintenance of the design and code through analysis, review and refactoring. Had the group firmly agreed on a better design from the beginning of the project, this could have been avoided.

6 The Overlay

The Overlay was an essential part of the application. It was needed to display various computational elements in their respective categories on the Obashi diagram. This was used when a user collected some cubes in the AR environment and then placed each cube in the correct category on the diagram.

This required thorough research and gathering of ideas from the whole team, in order to decide on the implementation. The implementation of the Overlay required many different approaches and subsequently testing each approach in Android Studio. This was necessary in order to find the most practical and visual appealing result.

The initial idea was displaying a pop up window after clicking a button [9]. Once clicked, a new layout would appear displaying the Obashi diagram, with the different categories as a dropdown list[1]. This approach was simple and straightforward, but lacked visual appeal along with some functionality. The issue here was once the cubes are placed in the correct category, the user would not be able to see them. To be able to visualize the collected elements in each category another layout would be needed. Alternatively, the cubes would have to be stored internally in some data structure. Storing them internally would not allow the user to see the cubes on the screen after they have placed them. This initial idea was the same idea as presented in the wireframes.

To combat this problem, another approach was used, which was based on moving from the AR Activity to a new Overlay Activity[5] and displaying the Obashi diagram inside the Overlay Activity. The full diagram was to be shown and the user, would be able to scroll in any direction and zoom in and out of the diagram [3]. After implementing this idea, the team soon realized that, again this was not the most stimulating design, since the full diagram on the screen was too small and the user would have to zoom in into each category to visualize the cubes. The transitioning from the Overlay activity back to the AR Activity was also not pleasant, as it would not save the cubes in the Overlay activity once exited. During the client meeting day 4, the customer made it clear that they preferred the overlay to be in the AR environment.

To overcome this issue, the team had to research more about, how to implement an Overlay inside the AR environment. The customers wanted to see the Overlay completed and working for the customer day 5. The solution found was fragments in android, which is an independent component that can be used by an activity. Multiple fragments can be combined in one activity, to build a multi-pane User Interface and reuse a fragment in multiple activities[2]. This allowed an overlay to pop up, after clicking a button inside the AR environment, with its own lifecycle and functionality. Customer day 5 approached and the overlay was not completed. Most of the time was spent getting familiar with how fragments work in android and fixing small bugs. For example, the overlay fragment was not displaying properly in AR or the overlay was not closing once clicked. The clients were still very happy with the progress, as the other functionalities were completed and working. After explaining the situation to the clients regarding the overlay, they were very understanding and wanted the

Overlay completed for the final presentation. Having not finished the Overlay put the team under some pressure, but after conducting the monthly retrospective, priorities were set and more team members started working on the overlay.

After the client meeting the main focus was the Overlay. The Overlay fragment has seven different categories from the Obashi diagram and each category has its own recyclerview[10]. These recyclerviews are created in the method inititateRecyclerViews in the file OverlayFragment.java, the ids[16] are defined in the overlay fragment xml file for each recyclerview. Each recycler view had their set of Cardviews displaying the text of the cubes for the diagram. Different colours were used for each category in the diagram to distinguish between them. The layout of the overlay was now completed, but the biggest hurdle was the linking of the Inventory and Overlay, in particular the placing of cubes from the inventory to the diagram and keeping track of the cubes. This problem took quite some time and effort by the team. The problem was overcome eventually with the following approach. Once a player chose a level, the elements which were scattered, appeared as an EMPTY box in the diagram. The other elements remained visible to the user. The scattered elements must first be collected in the inventory and then the player must select an element to place in the diagram. Once the player places the element in the correct position the EMPTY box is filled with the cube selected. This was possible due to the tag[14] every element has, whether its scattered or not. The elements in the inventory are also removed upon placing each cube in the diagram. All of this was achieved in the onInterceptTouchEvent method in the OverlayFragment.java file and the onBindViewHolder method in the OverlayRecyclerViewAdapter.java.

Overall, the Overlay was one of the most challenging parts of this application. It required multiple changes to be made and different approaches to be used, in order to find the most feasible outcome, which was user friendly, visually appealing and aligned with the customers expectations. Getting familiar with fragments and recyclerviews in Android Studio required more time than anticipated. In additon, had the team decided on the design from the start, this delay could have been avoided. The Overlay was left to the end, but this was inevitable as the other functionalities were needed beforehand, in order to link everything to the overlay.

7 Testing

When we set out to start writing tests for our application, we researched testing frameworks for AR applications but we realized that none were available. This is because testing an augmented reality application is quite a difficult task, due to the unpredictability and volatility of AR environments. Another big factor, is that AR environments are hugely “continuous”, it is very difficult to reach 100% test coverage when there are an almost infinite amount of states that an environment can be in. Moreover, AR applications are quite a new thing and a testing environment for them still hasn’t had time to be developed[15].

We conducted two distinct types of tests: instrumented tests and unit tests. In An-

droid, a unit test is just a regular JUnit test which runs on the hardware that an application is written on using the JVM or in a continuous integration environment. An instrumented test on the other hand, runs on a physical device or a device emulator which the app is designed for[8]. We tried to use unit tests in all of the cases possible, as they could be executed in the CI/CD pipeline.

In some cases such as the ArActivity, JUnit tests could not be used due to the Activity implementing a check for the version of OpenGL of the device that would close the application if it failed (we were not able to bypass this tests using mocking frameworks).

7.1 Unit Testing

After a lot of research, some frameworks were found, which helped build unit test. Mockito is a "unit based mocking framework"[4] which gave us an excellent alternative to simulation of an AR environment for testing purposes. Mockito was invaluable in the cases where we wanted to test methods which are called in an AR environment, and operate on data from this environment.

Instead of actually simulating an environment for our app to interact with, Mockito gave us the functionality to create "mock objects". A mock object is a dummy implementation of a class, where the output of its methods can be defined by us, the testers.

One example of our use of Mockito was when testing some methods that used the plane's dimensions, to scatter cubes. According to Google's ARCore developer references, a plane object "describes the current best knowledge of a real-world planar surface"[11]. Thus, a plane is an object which describes the plane that our app operates on. As it was unfeasible to create a plane using the provided AR Core API, we used Mockito to create a mock plane object that yielded the characteristics we needed for the tests[6].

This mock plane hugely simplified the testing process for any method which operated on a plane object, and also meant that tests were more consistent as we had the ability to define what is returned by Google's ARCore methods; methods which are otherwise volatile.

We also used Robolectric, which is a powerful alternative to conducting tests instrumentally (on an android device or emulator)[12]. We used it to test application interactions such as intent passing (the construct used to move between activities and pass information between them). These require the android environment to be configured in a sandbox in order have the appropriate context for the methods to operate on.

Using Robolectric allowed us to add these tests to the CI/CD pipeline and remove the manual work required to run instrumented tests directly on the device or the emulator.

7.2 Instrumented Testing

We had to test two activities using instrumented testing, due to the constraints stated above. For it we used Espresso, an Android testing framework specially designed for UI testing. With it we were able to capture intents and analyze their correctness[7].

8 Choice of Colours

The following diagrams (especially figure **??**) illustrate the process...

9 Managing Dress Sense

In this chapter, we describe how the implemented the system.

10 Kangaroo Practices

11 Knots and Bundles

12 Conclusions

Explain the wider lessons that you learned about software engineering, based on the specific issues discussed in previous sections. Reflect on the extent to which these lessons could be generalised to other types of software project. Relate the wider lessons to others reported in case studies in the software engineering literature.

References

[1] Anupam Chugh. Drop down list. https://www.journaldev.com/9231/android-spinner-drop-down-list#comments. Version updated on the 28/1/18.

[2] Anupam Chugh. Fragments. https://developer.android.com/guide/components/fragments. Version updated on the 27/12/19.

[3] Hariharan Developer. Zooming in and out. https://www.c-sharpcorner.com/article/zoom-view-in-android-using-android-studio/. Version updated on the 23/02/20.

[4] mockitoguy Et Al. Most popular mocking framework for unit tests written in java http://mockito.org. https://github.com/mockito/mockito. Version 1.x.

[5] Andy O'Sullivan. Moving between activities. https://appsandbiscuits.com/moving-between-activities-with-intents-android-8-697b3b70fdfd. Version updated on the 14/03/17.

[6] Google Android Reference. Build local unit tests. https://developer.android.com/training/testing/unit-testing/local-unit-tests. Version updated on the 27/12/19.

[7] Google Android Reference. Espresso. https://developer.android.com/training/testing/espresso. Version updated on the 27/12/19.

[8] Google Android Reference. Fundamentals of testing. https://developer.android.com/training/testing/fundamentals. Version updated on the 27/12/19.

[9] Google Android Reference. Pop up window. https://developer.android.com/reference/android/widget/PopupWindow. Version updated on the 27/12/19.

[10] Google Android Reference. Recyclerviews. https://developer.android.com/guide/topics/ui/layout/recyclerview. Version updated on the 27/12/19.

[11] Google ARCore reference. Plane. https://developers.google.com/ar/reference/java/arcore/reference/com/google/ar/core/Plane. Version updated on the 30/01/20.

[12] Roboelectric. Androidx test. http://robolectric.org/androidx_test/. Referenced on the 04/04/20.

[13] srowen Et Al. ZXing ("Zebra Crossing") barcode scanning library for Java, Android. https://github.com/zxing/zxing. Version 1.9.

[14] Android Teacher. Textview and tags. http://android4beginners.com/tag/textview-category/. Version updated on the 04/04/13.

[15] Tracy Watson. Specifics and challenges of augmented reality testing. https://skywell.software/blog/specifics-and-challenges-of-augmented-reality-testing/. Version updated on the 26/09/19.

[16] Jim White. Layout and id. https://www.intertech.com/Blog/android-layout-and-id-attribute/. Version updated on the 05/07/11.