University of Glasgow | School of Computing Science

Level 3 Project Case Study Dissertation

Flock Information Centre Mobile App: An Experience Report

Khena Dungu
Rawan Jeraq
Declan McBride
Kārlis Siders
Danielius Šukys

4th April 2022

Abstract

This experience report describes Team SE01's process in developing the Flock Information Centre mobile app as part of University of Glasgow's Year 3 Team Project course.

The report addresses the steps we took to plan, design, implement, test, and maintain an app on iOS and Android. It also addresses problems encountered, from using Flutter, Django and SQL to working in an agile development process, and our solutions to them.

The conclusion contains a summary of the general and wider lessons learnt throughout the software process.

1 Introduction

This paper presents a case study of mobile app development and the development of all accessories related to the app. As a project for a team of university students, it is also a case study of a team working for the first time with a real-world customer in an agile software development process. As such, the case study highlights how our team dealt with new technologies in a new environment while developing software with people we had not worked with before.

The rest of the case study is structured as follows. Section 2 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. The workflow of our team is introduced in section 3. Section 4 discusses our experience with the technologies we encountered. Section 5 focuses on the approach we took to handle the testing of our code. Then, section 6 explores our experience with our customer, while the team's experience with releasing an app to app stores is described in section 7. Finally, our conclusions and comparisons are demonstrated in section 8.

2 Case Study Background

Customer: Kim Hamer, School of Veterinary Medicine

2.1 Objective

The University of Glasgow School of Veterinary Medicine was looking for an app that is an extension of their website at https://kimhamer.wixsite.com/website/. The app share information on sheep farming with farmers to improve the health of their lambs and reduce the numbers of deaths in their flocks, thus improving animal and farmer welfare, farm viability and the environment. The goal was to convert this resource into an app that could be easily accessible on the iOS App Store and Google Play Store and would be a succinct resource for farmers. The mobile app should support offline use, have interactive elements and have the ability to be translated into multiple languages. The app should be designed in a way that it could be extended easily by the customer in the future.

2.2 System Requirements

We have split up the system requirements into three parts: general, standard user, and admin user requirements. The general requirements described how the mobile app should be built. This included the user-friendliness of the product, its availability on both Android and iOS platforms, and the ability for it to be translated in the future.

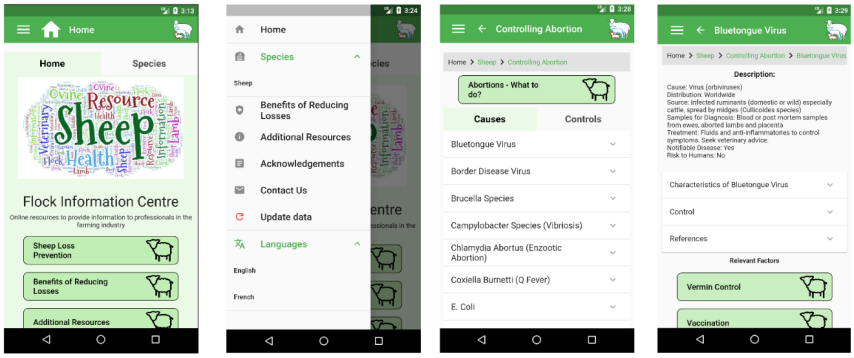

Figure 1: Screenshots of the mobile app on Android.

The standard user requirements described the functionality end-users should experience. It was critically important that the app could be used offline, and that it could be possible to download the newest updates once the network connection became available. Moreover, the user should be able to find information by narrowing down categories or by using a search bar. Finally, it was necessary to have a way to contact our customer, Kim, to report misinformation, suggest improvements.

Lastly, the admin requirements described how the customer of the project would be able to interact with a website to update the information on the app at any time. The key functionality was that the customer could verify themselves by logging in, then add/modify/delete various data shown on the mobile app, and once they were happy with the changes, release the new version of the data to users. Additionally, they would be able to see the feedback reports from the end-users.

2.3 Requirements Achieved

We have successfully built the app, and it is available now on the Apple App Store and the Google Play Store, as seen in Figure 1. A relational database has been built containing all of the data the customer wanted available to farmers. A web app has been developed using Django, which enables the customer to update the data at any time and send it directly to the users without having to release new builds of the app itself to the app stores.

3 Workflow

3.1 MoSCoW

Our project brief was broad and extensive. Therefore, we struggled to find a way to distil it into a list of the key priorities. Having just learnt about the MoSCoW method, we thought our project would be a good use case. Not only was it an understandable method for laymen to put features in a hierarchy of priorities, it has

also been successfully used in many other software development teams (Hatton 2008). After a small discussion, we decided this was the best method to go with as it allowed us to strictly separate our goals into 'Must have', 'Should have', and 'Could have' features. This process resulted in the following list:

'Must have' features:

- The app must have a user-friendly interface
- The app must have an information page for each category
- The customer must have an ability to add/modify/remove data
- The code needs to be prepared for potential translation

'Should have' features:

- The app should have a search bar
- The app should have a contact form

'Could have' features:

- The app could have a text-interpreting bot that would answer user questions

These priorities were discussed with our customer before work on the app started.

3.2 Communication

Throughout the entire process, our team agreed that communication must always be one of the highest priorities in our workflow. We started by creating multiple Teams channels for different purposes: a Customer channel for communication with our customer; a Developer channel for things related to development, including releases and issues of the project; a Coach channel for contacting our coach; and a Staff communication channel that was used for questions to course lecturers. Throughout the project, we noticed that such a structure has helped us with communication: we could talk to different stakeholders about issues relevant only to them in their channel and without bothering the rest. The level of organisation also helped the team to better make sense of the organisation of the whole project, which in turn helped in avoiding misunderstandings and errors. All of this considered, we have definitely benefited from maintaining this separation of interests in Teams channels, and we would use that in future projects.

Another important aspect of communication was our weekly in-person meetings. During our meetings we tried to maximise efficiency by helping each other with the issues we encountered in the previous week. Sometimes this involved working in sub-teams

on a complex problem, or occasionally we would use the pair-programming technique to get some perspective or seek a different approach from a colleague. During our weekly meetings, the chair (Khena) would set up a To-do list including tasks for every member of the team. After the New Year, we started focusing on To-do list deadlines a bit more to ensure we stuck to schedule and met our commitments. Having to-do lists week by week helped the team keep track of our priorities, and allowed us to split the sprint into smaller chunks to better understand how quickly we needed to do our tasks and plan accordingly.

We decided to split up into teams at the start of the project: front-end and back-end. This allowed us to work on different aspects of the project simultaneously. We could create complementing parts side by side, adding features whilst updating the other team on our progress. Communication between the two teams helped increase understanding by every member on the whole team.

The front-end team was responsible for user interface, user experience and the dynamic (meaning interface elements were based on the database content, rather than hardcoded) aspects of the app. Given this team's focus on the user experience and aesthetics of the app, they had a lot of communication with the customer to ensure the app looked the way they had envisioned it. This team would take the customer's feedback and make changes as necessary.

The back-end team was focused on modelling and building a SQLite database. The back-end team was also responsible for populating and preparing queries for the database. They had less interaction with the customer, but would still try to introduce them to the behind-the-scenes functionality of the app and convey the importance of it. The customer feedback was often more abstract for this team and was often in the form of questions, which this team worked hard to answer in an accessible way, which is reflected upon in section 6.

After the New Year we split up further, adding a web team that was responsible for working on the admin interface for the customer to be able to update the app data.

4 New Technologies

Most of the team had little or no experience with the technologies required to build a mobile app. Both front-end and back-end teams had to do their share of research and learning to successfully move forward with development.

4.1 Front-End Team

4.1.1 Framework

The first difficulty that was encountered by the front-end team was choosing the right technology. None of us had any experience making production-ready mobile

apps and only marginal experience making Android apps, we were unsure where to begin. The early discussions with the customer were leading us to develop Android-first and iOS-second. However, after a little research, we discovered the existence of app development frameworks which enable developers to build consistent apps between the two platforms. Our research showed that we could either use React Native or Flutter as they were the two most used frameworks in the industry. We decided to use Flutter as it has extensive support from its creator Google, including an optimised IDE (Android Studio). We felt this made it more beginner-friendly than React Native. It is also very lightweight, even compared to native applications (Olsson 2020). Thus, much of the start of the development process was about learning how to use the Flutter framework along with its in-built language, Dart.

4.1.2 Asynchronicity

Familiarising ourselves with the Flutter framework was another hard task involving lots of tutorials and documentation reading. A significant challenge for us was moving from single-thread applications to the complex asynchronous mechanics that are standard in a Flutter app. A lot of time was spent dealing with errors that occurred when the app was trying to access data that was not yet available. Generally, asynchronism was difficult to understand, visualise and test. The specific use case of our mobile application involves including asynchronous data access inside asynchronous access of other data. This is a unique requirement which meant that there were sometimes few or no developers with similar problems on websites such as Stack Overflow. This led to a lot of frustration and time spent experimenting with our coding environment to solve these problems. The experience has taught us a valuable lesson that even though one might have great experience in using similar technologies, there are always some underlying differences that might hinder progress and require careful reading of documentation and experimentation in order to solve these occurring problems.

4.1.3 External Packages

Another difficulty that arose was the unexpected behaviours of some packages that we used. One of them was a breadcrumb package. This package handled overflow (meaning interface elements would be generated off-screen) very well, but it did not handle navigation at all, other than having an `onTap()` function for each breadcrumb item. We tried looking at the source code of the package to see if there was something that we had missed in the package library, but there was no reference to navigation. We then tried to get in contact with the author of the package, but they did not respond. Thus, we had to solve this issue on our own, reflected upon in section 4.1.4.

Similarly, the *Flowder* package used to download the online database to the app was used because of its simple syntax, but the package had a known bug with pull requests to fix it being unaddressed for months, which caused the whole app to crash. In order to fix this bug we created a fork on GitHub with various fixes, which was the version

then used in the final app. From this experience we learnt that, although packages are assumed to be working, they may not always fit perfectly into your code.

The team encountered an app-breaking error when one of our packages was updated just days before code-freeze. This was simple to resolve when the issue was discovered, but it took several hours of work and frustration to figure out.

In the end, we learnt that using external packages is not the panacea that the team imagined it to be: sometimes they are not exactly what we might need, sometimes they might not have been tested well enough internally, and sometimes the expected fast solution might take a long time to verify and debug.

4.1.4 Breadcrumbs

Adding breadcrumbs for navigation took approximately 4 months for us to complete. The idea was to have a breadcrumb bar to show previous pages the user has visited, where clicking on a breadcrumb navigated you back to that page and cleared all breadcrumbs which followed from that point.

There were many stops on our breadcrumb journey. Some of the early attempts involved generating clickable ellipses in between the first and last breadcrumb to avoid overflow errors, and changing the `breadcrumbItem` to allow for individual navigation. This caused a lot of runtime errors that would occasionally cause the program to crash, revealing some fundamental issues in our implementation. As a result, we turned to the Flutter breadcrumb package instead of trying to reinvent the wheel with our own creation.

The team hit a plateau with this feature shortly after, with little progress being made. We reached a critical juncture in development, where our app was transitioning from "static" (where pages and information were hardcoded) to "dynamic" (where template pages were used to generate the GUI based on user selections and information from the database). We decided at this point to use the pair-programming technique. Pair-programming has been shown to increase success rates of tasks, improve satisfaction in development teams, and have an overall positive effect if implemented in a software team or university course (Mendes et al. 2005).

The pair in this case was Kārlis, who had been the lead on the transition from static to dynamic, and Rawan, who had been the lead on implementing the breadcrumb feature. This approach was successful and significantly sped up the process. Pair-programming allowed the pair to draw on each other's strengths and complement each other's gap in knowledge, which helped in understanding not just the task itself but also its context in the project in general.

Getting over the plateau revealed new errors, however. We realised that navigation only worked when you were navigating between different types of pages – for example, from an info page to a category page. This was because the page routes were still static and still were called `/infopage`, `/categorypage` even though pages were generated

and named dynamically. This meant that, when using `popUntil()` method used in Flutter navigation, the program did not recognise that varying instances of the same type of page were actually different pages.

Initially, we tried to make the routes dynamic to fix this issue, but it quickly became apparent that this was a much larger task than we expected as it required multiple changes to various aspects of the program. Doing so would not have been possible in the remaining development time. After ruling out that fix, we discovered yet another error with the navigation, wherein clicking on previous pages would take the user further back than they intended, occasionally even causing the program to crash. It was becoming clear that our code was messy and confusing. There was leftover unnecessary code from fixing other issues that had arisen, and from previous sprints. The solution required a fresh pair of eyes to question why certain code existed and clean up any unnecessary parts. To do this, we did pair-programming again and this, finally, fixed the breadcrumbs and ended a four-month task that we had anticipated to take about two weeks.

There were valuable lessons learnt throughout this long process. We learnt the importance of keeping our code clean. Adding features to already established code is fun, and it is easy to spend all your time on that, but we now know that time must also be allotted for improvement and maintenance of our code base. We also learnt just how valuable it can be to work collaboratively and harness each other's knowledge and strengths.

4.2 Back-End Team

4.2.1 Creating the Database

The back-end team also experienced some difficulties with Flutter, the main issue being the very concise official documentation. The official documentation code samples of database connection only included a database with one table, for example. As our project involved a large database with many tables this was not enough information to ensure we understood how to work with it. To combat this we had to turn to Stack Overflow to find a more realistic example of working with a database in Flutter. The documentation was also lacking of examples of connecting an existing database, instead describing only how to set up a new database in your code. It took a long time to find a reasonable explanation of how to do this and how to make it work with the front-end. Our main takeaway from this was that you cannot always rely on documentation to answer all of your questions, but often you are not the first one facing the issue, and a solution might still be found elsewhere.

4.2.2 Populating the Database

Finding a way to populate the database we created for the website brought its own set of challenges. The website we were given from our customer included all the data we

needed but importing this data manually was difficult and tedious. Our aim was to model the database to not only be flexible, but also include support for translations, this meant we ended up having a lot of foreign key linking. To make this process easier for us, we created a Python script that would take one CSV file, populate and then link the data automatically. The data connection was described by 6 headers in the CSV file:

ID Section Title Section Data Section Type Parent Sections Related Sections

Executing the Python script allowed for inserting all data, and connecting one section to another as both parent-child and two-sided relationships. Through this we learnt that we can usually find a more efficient solution when you give an issue a bit more thought.

4.3 Site to Edit the Database

Building a website for editing content was a great way to improve Django skills learnt in the WAD2 course last year. The most challenging parts included the deployment of the website and handling multiple databases at once.

For deployment, we chose to use Docker to make the website easily reproducible on any operating system (Bucchiarone et al. 2018) (Knoche & Eichelberger 2018). We used Docker Compose to create the database and website containers and also execute all commands needed for set-up. The result is that the website can be set up using two docker commands: "docker-compose build" and "docker-compose up -d". This ensures that the server is up and running, the user database is migrated, and a superuser is created (custom credentials based on the environment file). We think that the experience of deploying a website using Docker is extremely useful. Docker is a highly-valuable tool that is often used in practice, hence, it is a good addition to the skill set of a software developer.

Another complexity of the website was managing two databases at the same time. One database, the content database, is a complete SQLite replica of the database used in the Flutter application - it contains the information about lamb health and all other content. This database is modified by the user and then exported as an update to the Flutter application. However, the website also needed an authentication mechanism for our customer. Since we did not want database updates to include user data, we decided to have another PostgreSQL database, the user database, specifically for user storage. It was exciting to learn to manage two databases at once as it was a problem we had never encountered before.

As the content of the database was generated and pre-populated via a script, it did not work well with Django models. Hence, all the queries needed to be written via a custom SQL query, and some of the queries were quite complex. They included multiple joins and intermediate table creations. This meant we had to spend more time on SQL to better understand how it works and how to write a query that behaves exactly the way it needs to behave. The creation of the website was a good

deep-dive into the technologies we have learnt at the university, allowing us to better understand how they work behind the scenes.

5 Pipeline and Testing

The pipeline was a very difficult aspect of the project to figure out by ourselves as we did not have any experience building pipelines for any version control system before. As such, we had to learn about what GitLab runners are, how to set them up with correct syntax, and most importantly, how to interact with Flutter in the pipeline. We started with the ambitious goals of having tests (both unit and static analysis), building the app and publishing to the respective app stores. We researched for many hours, using online tutorials and documentation to figure out how to do this with the shared GitLab runners the team was allocated; however, there is little online about this specific combination of Flutter and shared runners. Thus, we settled on only running unit tests as that seemed to be the most important aspect and the rest was done manually by the team. Even though having continuous integration and deployment has many benefits, including improving the product's testing and predictability and positively impacting the team's communication and productivity (Ståhl & Bosch 2013), the team felt that it was reaching a tipping point in the balance between time spent optimising the software, i.e., building the pipeline, and time spent working on adding, fixing and maintaining new features that could be demonstrated to the customer as successful tasks at the end of the sprint. With more time, the team would definitely extend the pipeline's functionality.

The team implemented the series of tests using the Flutter_test package. Flutter_test is built specifically to test that each element of an app is building and displaying correctly. Flutter is largely an interface-first paradigm, this makes it difficult to implement standard unit tests extensively. Each interface element in Flutter is called a widget. As well as being responsible for displaying pieces of the interface, the widget also handles computation. Flutter_test checks that the widget looks how it is supposed to, checking the text it displays, or that it generates the correct child widgets. It also makes sure that the computation works correctly. All in all 8 Widget Tests were implemented, looking at 14 different crucial widgets. Our tests only pass if our main 6 elements are working: the menu bar, hamburger menu, the interactive buttons, the tab bars, and the drop-down menus.

6 Customer Management Experience

The customer management experience was one of the most significant learning curves throughout the project. We learnt a lot of technical things about how best to interact with a customer, including keeping meetings organised, short, and to the point. As software developers, we have learnt the importance of the communication with the customer and the problems that might arise when developers and the customer are not on the same page.

Our customer was very helpful in terms of providing us with necessary data for our project. They sent us various sources including slides, a mind map and an existing website. These allowed us to begin the development process, but, as it went on, we found there were a few complications. As there was a lot of data given, it felt at times as though there was a contradiction between the different sources. We addressed this issue by organising extra meetings with the customer and also just writing questions to the customer directly in the MS Teams Customer channel.

An unexpected but greatly appreciated benefit from the project was the possibility for the developers to visit a local sheep farm with the customer and a representative from the other team working with the same customer. From this experience the team gained not only a large collection of sheep photographs, but also valuable feedback from a farmer, the ultimate end-user of the app we had developed. One piece of their feedback was that, given the fact that a large percentage of farmers in the United Kingdom have dyslexia (Smith et al. 2016), it would be appreciated if the app did not display large blocks of text without some kind of summary. With that feedback in mind, a discussion was had with the customer to edit the content of the information page, while the UI itself was changed to include expandable headings on top of each section of a block of text.

The one thing we had to understand throughout the development was that software engineers and customers are often in two different worlds and the connection between those words is an important one to establish. The customers can only see a small part of the total effort that is put into a project. This often results in the customer over-emphasising easily changeable things and not giving enough attention to the things that are the building blocks of a project. Therefore, it is a developer's job to ensure that the right questions are asked to get the necessary feedback. It is also a developer's job to communicate that a lot of the effort is put to things that are not seen to the customer and happen in the background. Software engineers need to convey that these background tasks are particularly important ones and sufficient time needs to be spent on them.

7 Releasing The App

No member of the team had any previous experience with releasing an app to any app store, and we were surprised at how complicated the whole process is. The release form felt endless, there were many questions we did not fully understand, and we also encountered some problems with exporting the application. Some of the documentation and answers to questions available on the Internet were outdated and contradictory. For example, the necessary changes to the `gradle.build` file were described differently from what was actually required and most of the solutions found online did not work.

As Flutter is developed and maintained by Google, the main source of difficulty in releasing the Android app was getting through the bureaucracy. The iOS release of the app was more difficult as the iOS build is not handled solely by the IDE in the

way it is for Android. The team had to create a large suite of icons in assorted sizes, had to handle juggling errors from two different IDEs and spent a significant amount of time dealing with obscure and hard to understand build errors in XCode. The app was also rejected from the Apple App Store initially, which was a great learning experience, if very frustrating.

The team has learnt that build settings are critical and should be set up correctly from the beginning. In the future, if any member of the team were to do a similar project, agreeing on minimum OS requirements and setting up build profiles for release and debugging would all happen at the beginning of the development process. Additionally, the various icons that are required would have been developed at the design and prototyping stage. Agreeing a version-numbering system would be part of initial planning, not something decided at build time. Ensuring that the team has access to all necessary emulators to gather the various screenshots that the app stores require would be done during the initial setup of development environments.

The team now know that releasing is a difficult, precise and finicky process that requires the same amount of planning as any other aspect of the project. Therefore, we agreed that, if there was more time for development, part of that time would be invested into automated deployment, which would decrease continuous work for the developers, make deployment much faster and also increase the reliability of each release as there would be less room for human error (Luhana et al. 2018).

8 Conclusions

8.1 Summary

In short, the team achieved all the required goals in making an offline, translatable, UI-friendly, and extensible mobile app that is published in both big mobile app stores. Although the team had to learn and get accustomed to both the Agile workflow with a real customer and new technologies, including Flutter and Dart, SQL database population, pipelines, and releasing an app on two separate platforms, we persevered and finished the project wiser and better software developers than when we started.

8.2 Main Themes

One of the main themes throughout the development process has been the amount of new technologies, practices, and concepts each member of our team has had to learn. This is much easier to do in a team environment where anyone can ask for help, collaborate and exchange knowledge in order to improve. The agile methods implemented by the team, e.g., retrospectives, pair-programming, etc., helped us accomplish tasks that we did not feel entirely confident in even starting.

8.3 Key Takeaways

In general, the team struggled and prevailed in making a usable and extensible mobile app by working collaboratively, communicating everyone's desires and workload, and delegating tasks according to our strengths. The key takeaway each and every member of the team leaves the project with is the experience of creating a real product in a real production environment and all its additions, like customer meetings, software process methodologies, learning the intricacies of Git, the whole software process of design, implementation and release, and communicating with a diverse team.

References

Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S. T. & Mazzara, M. (2018), 'From monolithic to microservices: An experience report from the banking domain', *IEEE Software* **35**(3), 50–55.

Hatton, S. (2008), Choosing the right prioritisation method, *in* '19th Australian Conference on Software Engineering (aswec 2008)', pp. 517–526.

Knoche, H. & Eichelberger, H. (2018), Using the raspberry pi and docker for replicable performance experiments: Experience paper, *in* 'Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering', ICPE '18, Association for Computing Machinery, New York, NY, USA, p. 305–316.
URL: *https://doi.org/10.1145/3184407.3184431*

Luhana, K. K., Schindler, C. & Slany, W. (2018), Streamlining mobile app deployment with jenkins and fastlane in the case of catrobat's pocket code, *in* '2018 IEEE International Conference on Innovative Research and Development (ICIRD)', pp. 1–6.

Mendes, E., Al-Fakhri, L. B. & Luxton-Reilly, A. (2005), Investigating pair-programming in a 2nd-year software development and design computer science course, *in* 'Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education', ITiCSE '05, Association for Computing Machinery, New York, NY, USA, p. 296–300.
URL: *https://doi.org/10.1145/1067445.1067526*

Olsson, M. (2020), A comparison of performance and looks between flutter and native applications, Master's thesis, Blekinge Institute of Technology.

Smith, R., Conley, G., Smith, A. M. & McElwee, G. (2016), 'Assessing the impact of 'farming with dyslexia' on local rural economies', *Local Economy* **31**(5), 529–538.
URL: *https://doi.org/10.1177/0269094216655404*

Ståhl, D. & Bosch, J. (2013), Experienced benefits of continuous integration in industry software product development: A case study, *in* 'The 12th IASTED International Conference on Software Engineering', pp. 736–743.