On The Development of a Mobile Survey Framework with Dynamic Code Loading

Nikola Toljić

Dissertation 2014

Erasmus Mundus MSc in Dependable Software Systems

NUI MAYNOOTH

Ollscoil na hÉireann Má Nuad

Department of Computer Science

National University of Ireland, Maynooth

Co. Kildare, Ireland

A dissertation submitted in partial fulfilment

of the requirements for the

Erasmus Mundus MSc Dependable Software Systems

Head of Department : Dr Adam Winstanley

Supervisor : Dr Charles Markham

13th of June 2014

European Commission

ERASMUS MUNDUS

Abstract

Conducting research based on surveys and experiment is a laborious task. This report describes the design and implementation of a system that aims to reduce some of the workload researchers are facing when gathering data from the general population. This is achieved by creating a survey framework which allows the creation and distribution of surveys, as well as the collection of surveys results. In addition to textual questions, the framework allows the researcher to include coded experiments which are distributed with the survey and loaded by the framework at runtime. The survey tool, implemented on the Android™ platform, supports Android-based code modules but also web content and Unity code modules. Interfaces to facilitate communication between external code modules and the survey tool are in place for each of the three supported module types.

The generic nature of the framework coupled with its modularity and the ability to work without a permanent network connection make is suitable for a vast number of research scenarios. To show the feasibility of the system, a survey investigating the correlation between cognitive function and driver behaviour is conducted. It shows the level of automation that can be achieved and, simultaneously validates the system. System tests were used to verify the framework.

Category: D.2.13 [**Reusable Software**]: *Reusable libraries*

Terms: Management, Design, Reliability

Keywords: Android, Survey Framework, Survey Tool, Code injection, External modules, Dynamic Code Loading, Cognitive function, Driver behaviour

Acknowledgements

First, I want to thank my supervisor Dr Charles Markham for the continued support throughout the project. It was a pleasure working with him and I am grateful for his advice and guidance.

Second, I would like to thank Dr Seán Commins who gave invaluable insights into the psychological aspects of the research domain.

My Erasmus Mundus MSc in Dependable Software Systems and therefore also this project were funded in part by the Education, Audiovisual and Culture Executive Agency.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project is 21,545 words long, including project specification and plan.

In submitting this project report to the National University of Ireland, Maynooth, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web.

I retain the copyright in this work.

Nikola Toljić

Table of Figures

Table of Contents

1 Introduction

How does a person's reaction time influence their behaviour as a driver? Is there a link between a driver's cognitive abilities and their behaviour on the road, like their perception of risk? Do mentally capable people react differently to challenging traffic situations, compared to people whose mental capabilities are impaired? Answers to those and many more questions is what *SimRG*, a research group at the National University of Ireland, Maynooth are trying to find. The main tool for their research are driving simulators, the kind of simulator that does not take up an entire building, but a corner in a room within the department nonetheless. This requires research subjects to come into the lab to partake in one or more of the experiments.

Across departments and organisations, as well as research areas this is still the prevalent modus operandi when it comes to research conducted where people are the main subject. The approach of bringing people to the researchers, when not necessary, severely limits the potential meaningfulness of the research results. Having research subjects come to the laboratory where the experiments are conducted sometimes cannot be avoided. After all, it would be logistically impractical to move bulky medical equipment such as MRI (magnetic resonance imaging) machines. In many other cases however, this restriction does not apply. Rather, the technology used by the researchers ties them to a specific location which is not necessarily dictated by the nature of the experiment.

The benefit of being able to acquire data from a bigger set of research subjects is manifold. Apart from a higher statistical relevance, more people generally also leads to a wider demographic, which could lead to a more granular answer to the posed research question.

The challenge however, is to increasing the research's reach without compromising its quality. After all, for many researchers the laboratory is an environment they control, allowing them to prepare it for their experiment and to be sure that there is no external factors influencing the results.

1.1 Problem & Motivation

This project attempts to solve the problem of increasing the number of participants in research surveys without a negative impact on experiment procedure.

In particular we focus on cases where the research subject simply answers a pen and paper based survey or participates in computer aided experiments that do not require specialised hardware; since these are the examples where there is no benefit to restricting the research to a particular location. Our proposed solution focuses on a combination of mobile and web technologies, which when brought together, provided an end-to-end tool chain for researchers to create their survey, deliver it to participants and collect the results. The minimum requirement for a researcher to use the system is a mobile device running on the Android™ operating system and access to a web browser. The system that was designed and implemented allows researchers to use predefined question types or to implement and add their own question types and experiments. Third party modules are loaded at runtime and can be HTML, Android or Unity based. This gives researchers a wide variety of options some of which do not require them to know anything about the Android operating system, thereby eliminating the entry barrier they would otherwise face.

By building our system for Android, we take advantage of the inherent mobility of smartphones and tablet devices to bring the experiment to the participant rather than the other way around. This not only enables researchers to survey more people of the same demographic group as before, but also gives them access to parts of the society that are very difficult to accommodate in a laboratory environment. A prime example is patients confined to hospital that can provide data of great relevance to the research conducted; this is achievable by using a tablet device and bringing it into the hospital and to the patients.

At the core, the system is designed to allow participants to participate in experiments without supervision. This fact can be taken advantage of to conduct research beyond geographic boundaries by using people's own smart devices, which have become common place in most households in the developed world. However, this depends on the actual experiment that is being conducted.

Based on the above information, two technical research questions for this work were identified.

TQ1: How can Android and web technologies be leveraged to build a general purpose survey framework?

TQ2: How can the system execute remote code without compromising the device's security?

In addition, a psychological experiment is conducted to prove the viability of the system implemented, for which the following research question was identified.

PQ1: Is there a correlation in user performance between the Stroop test and a vehicle following experiment?

The remainder of this document is structured as follows. Related work to the proposed research questions is introduced in chapter 2. Chapter 3 gives an overview of the development approach taken to implement the system. Question TQ1 is answered in chapter 4 and 5, with a subsection in chapter 5 being dedicated to question TQ2. The setup and results of the work relating to question PQ1 are discussed in chapter 6, followed by a general system evaluation in chapter 7. Finally, the document describes future work and gives a conclusion in chapter 8 and 9, respectively.

2 Related Work

Creating automated systems to replace pen and paper based survey forms and questionnaires is hardly a new idea. Neither is deploying such a system on a mobile platform. Loading execution binaries at run time is also a well-established feature in many programming languages and its security implications have been discussed in various academic reports. This section highlights the academic work and commercial products and services in those areas and shows how they are related to the system discussed in this report.

2.1 Survey tools and services

Long gone are the times where researchers are forced to work with pen and paper as their only tools to collect information from survey participants. Electronic survey frameworks come in many forms and from many different providers. Generally, they can be split into two categories, online and offline frameworks.

2.1.1 Online survey frameworks

Some well-known online survey frameworks are SurveyMonkey, KwikSurveys and Google's consumer surveys [1] [2][3]. Those online services provide a web based user interface for both, the authoring of surveys and the survey delivery and response. In addition to market researcher and analysts, the academic community is also interested in the possible applications of online survey tools. Pargas et al. proposes an authoring tool for dynamic online surveys and Burkey and Kuechler are investigating web-based surveys in the context of corporate information gathering [4][5]. A more general accumulation of knowledge on online surveys is presented in Singh et al. [6]. Looking at online surveys from a slightly different perspective are Kite and Soh, by basing it on calendar events [7].

Looking at question/answer systems in general, there is also work being done on the Android platform, where Atterwala et al. are analysing how mobile platforms can be used for market research [8]. In Stradiotto et al. an e-voting system is proposed using the Android platform [9]. The main drawback of the aforementioned solutions is that they all require a permanent network connection to function properly.

2.1.2 Offline survey frameworks

The second big category is offline frameworks. Since, by definition, these frameworks operate locally, offline frameworks cannot rely on a web browser but rather require a native application to present a user interface to the participant. When researching this area it quickly becomes apparent that offline tools rely heavily on the portability of the devices they run on.
The Google Play store is the main distribution point for Android application and contains a plethora of offline survey tools, including popular examples such as Dooblo, Rollapoll and SurveyPocket [10][11][12]. A brief analysis indicates that none of the available tools offer a feature list comparable to the proposed system in this report.

But of course Android is not the only platform that has been used to create mobile survey frameworks, with QuickTapSurvey being an example of an iOS application [13].

2.1.3 Comparison to the new framework

Similar to the existing applications, the system proposed in this report aims at supporting offline surveys. However, there is a key difference to the existing services and tools. Allowing researchers to include their own experiments directly into the survey is a feature that was not present in any of the system that were analysed. While it is not in the scope of this project to conduct an exhaustive search for all existing survey tool the author believe that the support for third party code is a novel feature in the context of survey tools and services.

2.2 Code injection

While the idea of providing researchers with the ability to run experiments as part of their survey is novel, loading external code binaries into an application is not. In fact, among the many public Java APIs is the `ClassLoader` class, which facilities external code to be loaded dynamically. Loading external classes however, does pose a security risk [14]. Even though other ways of dynamic code loading could be used, the `ClassLoader` API seems to be the most popular approach within the academic setting. Class loading in the context of mobile devices is not a mechanism that is exclusive to the desktop environment, but is also used in the mobile domain [15]. Due to the inherent risks external code loading poses, its security implication has been widely discussed. Several approaches for implementing countermeasures to minimise the risk of remote code execution have been discussed in the academic community. The approaches range from automatic defence mechanisms as proposed in and to general strategies of loading untrusted programs [16][17][18]. Especially interesting is the analysis pertaining the Android platform performed by Poeplau et al., which is highly informative and recommended for the interested reader [19]. In similar research projects, the aim was to identify potential security holes in mobile applications [20][21]. Furthermore, Hatwar and Shelke propose a system to detect malicious dynamic code in Android applications [22]. In general, security seems to be a main concern in the mobile application space, as a variety of papers focus on this issue [23].

2.3 Android applications

Developing research tools for mobile application to increase the mobility and flexibility of researchers and their work is not a novel idea. There is general interest within the research community to include the Android platform into their toolbox [24].

Ample literature exists that highlights the benefits of mobile over stationary devices. Smartphones and tablets have an additional advantage in their portability over the classical portable device, the laptop. Other advantages are the slew of sensors integrated into smartphones and tables, not typically found in laptops or netbooks, such as GPS, Gyroscopes, Accelerometers and Step counters. Murphy and DiMarzio give a good introduction into mobile application development in general and the capabilities of the Android framework in particular [25][26].

In addition to devices running Android natively, several solutions exist that attempt to allow the execution of Android applications on other operating systems as well. While the Android SDK comes with the facility to run a virtual Android device using, its performance has often been criticised. As the platform and the SDK matured, the emulator saw some improvements as well. One of the probably most critical features increasing performance are the snapshot features, which allows a snapshot of the emulator to be taken to avoid having to boot the Android operating system within the emulator every time the emulator is started. Unfortunately, snapshots do not improve overall performance of the virtual device. The addition on GPU support however, greatly improved the performance. A detailed description on the use and the restrictions of the emulator can be found online at the official Android developer website [27].

Other tools to run Android on other platforms include Windroy and Bluestacks [28][29]. Both tools are windows applications that run an instance of a virtual android device, allowing the installation and execution of Android application. Another way of achieving this would be to install Android on a virtual machine. The drawback to all of these approaches is however, that the hardware the Android instance is running on usually does not offer the type of sensors present on most modern Android devices. Therefore, emulated devices can only be used to do limited testing and to run applications that do not require special hardware not available on the host machine. The current implementation of the proposed system was not tested in any virtual environment. However, since the framework itself does not require any specialised hardware sensors, it would be possible to run simple surveys within a virtual environment.

3 Development approach

This chapter gives an overview over the development techniques used in designing, implementing and testing the survey framework system. Furthermore, it describes the tools that aided the development process. Great care was taken to ensure the reproducibility and the traceability of all steps in the process. Especially any data processing that might have been accomplished using manual spreadsheet software was done in an automated fashion. This ensures that, given the same input data, anyone can produce the same output as described throughout this document.

The system was developed using an incremental approach. Due to the fact that not all requirements were known at the start of the project and that some domain as well as technical knowledge had to be acquired, a prototyping phase preceded the actual development phase. The prototyping phase was used to determine which functionality could be incorporated into the system envisioned. More importantly however, it also provided insight into the different technologies and frameworks and their interoperability.

The commitment to Android as the platform on which the core of the system would run was made in part because of the author's previous experience with the platform. The main reason it was chosen over the other widely popular operating system, iOS, was the fact that iOS simply would not allow for many of the features of the system to be implemented.
The commitment to Android subsequently lead to Eclipse (Kepler 4.3.1) being chosen as the main IDE (integrated development environment) due to the availability of the Android Development Tool (ADT 22.6.1) plugin for it.

Other technologies, programming and scripting languages and tools were chosen as their necessity became apparent. Due constraints on time, adoption time of alternative tools and technologies was one of the selection criteria for the inclusion into the project.

3 Development approach

Table 1 gives an overview of which technologies and tools were used throughout the project. It also shows for which purpose they were used.

Table 1: List of technologies and tools employed during the development of the project.

Technology/ Platform/Tool	Purpose
Android	Operating system for which the main application was developed.
Eclipse	IDE for the development of the Android components.
Notepad++	Used as the default text editor when Eclipse was not appropriate.
SQLite	The database technology used within the mobile application.
PostgreSQL	The database technology used on the web server.
Apache Webserver	Used to run the web server.
HTML, CSS	Used to generate the survey generation form UI.
Javascript, jQuery	Used to add the functionality to the survey generation form.
PHP	Used to add survey data processing capabilities to the server.
Unity	IDE to develop the Unity modules.
Monodevelop	IDE provided with Unity to develop the scripts for Unity.
Filezilla	Provides an easy interface to transfer files from and to the server.
pgAdmin III	Tool to administrate the PostgreSQL database on the server.
R, RStudio	Used for data analysis of the experiment results.
MS Excel	Used for data analysis of the experiment results.

3.1 Project scope

This section gives a brief overview of the work that was put into the creation of the system. Table 2 shows how many lines of code each component of the system has.

Table 2: A survey framework encompasses a total of 5267 lines of code (including comments). This table shows how extensive the work was for the individual parts of the project. With 4153 lines of code (LOC), the Android application holds the lion's share and is undoubtedly the most complex part of the system.

Technology/Language	LOC
Android application & modules	4153
Unity module	348
Web interface & modules	430
Java - data processing	308
R - data processing & analysis	28

Since the Android application and modules made up the bulk of the coding effort, more detailed on this part is shown in Table 3.

Table 3: The individual Android components that were developed as part of the survey framework. Instead of lines of code, the McCabe Cyclomatic Complexity is shown [30].

Android sub project	McCabe CC (Average)	Description
ResearchSurveyToolLib	1.444	Contains the core classes for third party modules
ResearchSurveytool	1.278	The main Android application.
StroopModule	1.765	The modules containing the Stroop task
DrivingModule	1.000	The Android part of the Unity module

The tables in this section give an indication as to how much work it took to create the proposed system and how complex the Android based modules are. The quantity described in Table 2 and the complexity in Table 3 are a product of the time invested into the implementation of the system.

4 System overview

This chapter gives an overview of the research survey tool created. The purpose of this tool is to provide a start-to-end solution for researchers who want to conduct surveys and/or experiments on a potentially large scale. From the design of the survey, its questions and experiment components, distribution and data collection, to providing the data in a central location for the researcher to access, the system created covers all steps necessary to conduct and evaluate a survey. The novelty in the work lies within the capability of the system to serve as a general purpose survey tool that can load third party code modules to extend and customise its functionality.

In order to understand such a relatively complex system, we will first introduce the components and show how they interact with each other. Subsequently, the individual components and their capabilities will be described in more detail. Figure 1 shows the high level tasks in a work flow diagram, showing the order in which they are executed and how they rely on each other's results.

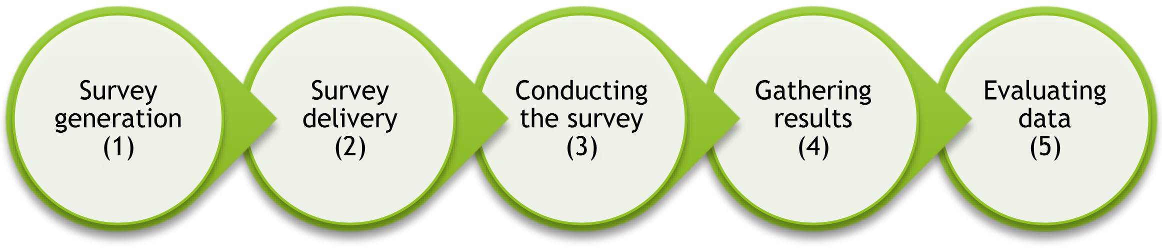
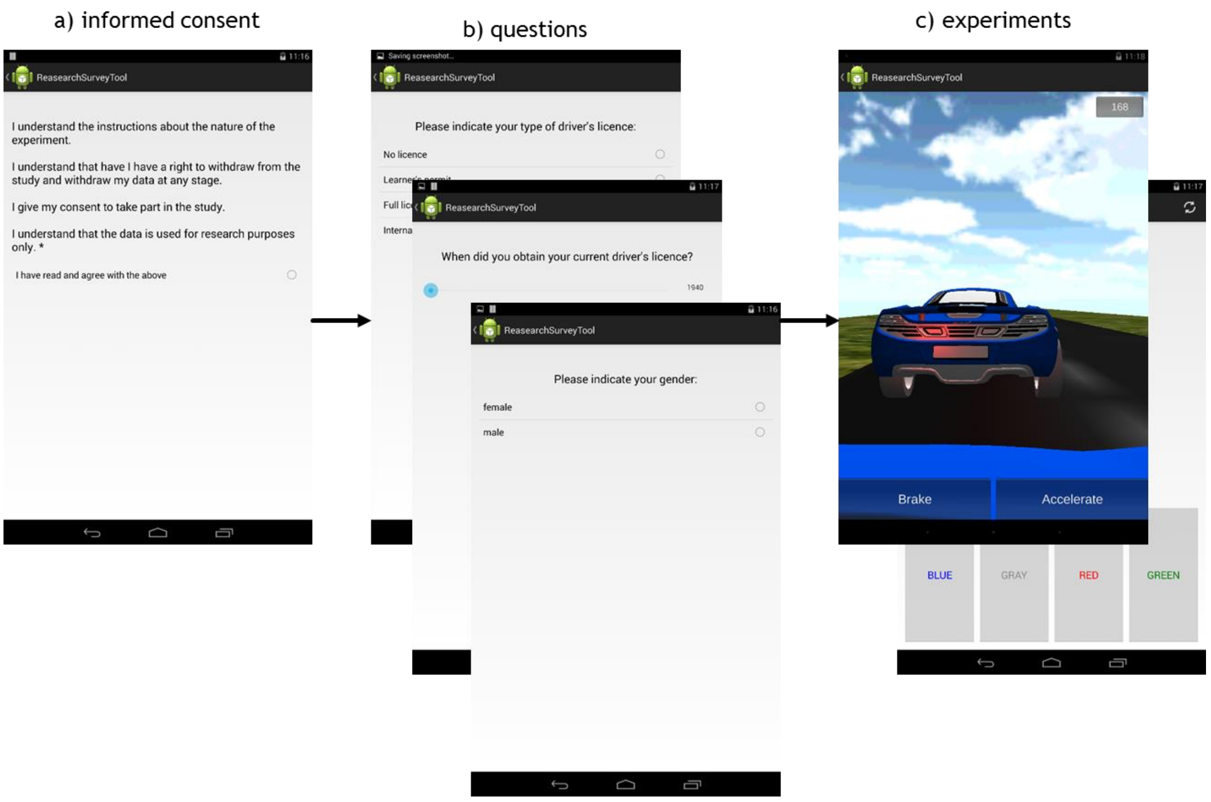


Figure 1: Survey process: These are the steps researchers need to perform when conducting a survey.

As can be seen, the first logical step, once the idea for the survey exists, is to create the survey. This task encompasses determining whether any experiments are required or whether just textual responses will suffice, phrasing the questions and choosing how the survey participants should answer the question (e.g. open questions, multiple choice, etc.). The next step requires a delivery mechanism to bring the survey to the survey participants. In a traditional setting, the researchers would bring a printed copy of the survey questions to the participants. More and more common is sending surveys to participants with links so that they can be answered in a web browser [1][2]. Once the participants receive the survey they need to answer the questions and/or perform the required experiment tasks. This could be anything from simply ticking boxes with pen and paper to recording a video of the participant performing a task. Step number four involves gathering the survey results, which can be written responses or recorded experiment data. Only after all those steps are successfully accomplished the researchers can begin their actual work of evaluating and analysing the data gathered. One of the goals of this project was to automate the first four steps so as to significantly reduce the effort researchers need to invest before they can get to the data they need to do the survey generation and survey delivery.

To achieve the first two tasks (survey generation and survey delivery), web technologies are utilised to provide global access to the system. A straightforward web interface was created using HTML, CSS, JavaScript and PHP. It is accessible through any modern web browser and allows the creation of surveys in a matter of minutes. Once all the required information is entered, the survey is stored on the server and the researcher is provided with a link which can be used to deliver the survey to the participants. Researchers can either distribute the link to participants on paper or via email or social media platforms. Since participants cannot use the link on its own, they need a way to respond to the survey questions. Therefore, a mobile application was created. The reasons for choosing a mobile platform over the conventional desktop environment are threefold. Firstly, people are spending more and more time with their mobile devices in comparison to their desktop computers or laptops [31][32]. Secondly, using mobile devices adds to the flexibility of the overall system. Rather than having to bring people into a laboratory to a dedicated survey machine, researchers can easily bring the survey to the participants. This is especially valuable in scenarios where researchers deal with people with limited mobility. The third reason for going mobile is that people already spent a significant time interacting with their mobile devices, which means that they are already familiar with the survey device itself. The reason for choosing the Android platform over other platforms is twofold, the authors experience with the platform on one hand, and the sheer ubiquity of devices running the platform on the other hand [33]. Furthermore, the open nature of the Android operating system enables relatively straightforward code injection mechanisms. Choosing a different operating system would severely limit the possible reach of the framework. Moreover, both Apple's iOS and Mircosoft's Windows Phone do not support dynamic code loading, which is a key piece of functionality of the proposed system. Figure 2 shows the interface the survey participant interacts with.



I understand the instructions about the nature of the experiment.

I understand that have I have a right to withdraw from the study and withdraw my data at any stage.

I give my consent to take part in the study.

I understand that the data is used for research purposes only. *

I have read and agree with the above

Please indicate your type of driver's licence:

When did you obtain your current driver's licence?

Please indicate your gender:

Figure 2: Typical format of a survey on a mobile device. First, the participant has to give informed consent. They are then presented with a set of questions, followed by any experiments the survey might contain. The order of questions and experiments does not have to follow this particular format. The informed consent is however, always the first screen the participants are shown.

In addition to serving as a survey delivery mechanism the mobile application also facilitates the execution of tasks three and four (conducting the survey and gathering the results), see Figure 1. Participants are provided with a simple and intuitive interface to answer survey questions and perform experiment tasks. Once participants have successfully answered all required questions and performed tasks, the data is stored on the device. It is then at the discretion of the participant to submit the data by uploading it to the server. The server also provides access to the survey generation web interface.

Once the data is gathered, the researchers can download the aggregated data as comma separated values (csv file), with experimental data embedded in JSON format. Alternatively, a custom format, such as comma separated values, can be used instead of JSON. The specific format depends on if or how the data is processed at a later stage. This allows them to edit the data in any spreadsheet program that can handle this file format. Currently, this is the only way to access the data. Providing a web interface to directly access data from the database is discussed in Chapter 8.

It is important to note that the mobile application is designed so that multiple participants can use a single device, allowing the use case, where the researcher provides access to one device. Alternatively it would also be possible that participants install and use the application on their own device. The system model was designed with the latter scenario in mind. The implementation of the proof of concept application favours the former user scenario, and does not currently support multiple devices to be used for the same survey. This distinction is irrelevant for the underlying concepts of the framework and can therefore be ignored. Wherever this report discusses implementation details that deviate from the original design, this will be specifically mentioned.

4.1 System model

Before going into detail on the functionality of the individual system components, this section explains in detail the underlying model of the system. By first showing the use cases and scenarios we want our system to support, we can then make the case for various design decisions that were made with regards to the system model.

The bare minimum that our model was to support is simple questionnaires. Presenting the participant with questions and giving them the possibility to answer them is at the very core of the model. Different types of questions require answers in different formats. Our model therefore categorises questions based on the expected/desired response. Specifically, the questions types as seen in Table 4 below have been identified.

Table 4: Question types supported by the survey framework.

Question Type	Expected response	Example
Open question	Unstructured textual information	How do you see your future?
Range question	Single value within a predefined range	Please state your age
Single select question	Single value picked from a set of predefined options	Which of the following attributes describes your character best?
Multi select question	Any number of values picked from a set predefined options	Which of the following attributes apply to you?
Location question	A pair of longitude/latitude values	Where do you live?

Note that the question types focus solely on the structure of the expected response, not its content. The open question type, for example, could be split up into sub types depending on the nature of the information, e.g. alphabetical, numerical, alphanumerical etc. However, while this might improve the user experience in some scenarios, we wanted to keep the basic model independent of any particular delivery mechanism (i.e. mouse & keyboard vs. touch screen).

Another feature we want our model to support is question grouping. Question grouping allows related questions to be grouped together in a logical unit. Researcher can use it to create different question groups for different research questions they are trying to answer and still provide them in one single survey. Moreover, the model was to support different types of question groups. In addition to standard, one time question groups, recurring questions are supported as well. This allows for studies to be conducted such as mood studies, where participants are asked the same question over a period of time in predetermined time intervals. Moreover, the model allows for question groups to be marked as active, which means that the survey participant will receive a reminder to answer this specific set of questions. While the first use case, with a single set of questions which are to be answered only once, does not require any form of user identification, recurring questions require that a user can be re-identified so that their responses can be aggregated properly. This can be achieved by one of two ways. The first is to explicitly require users to identify themselves by providing some login mechanism. This would not only create an additional engagement barrier for the participant but it would also jeopardise the participant's trust of their anonymity. The second approach implicitly guarantees re-identification, which is to have one device running its independent instance of the system for each participant. While this approach might sound very cost-inefficient, the idea is to make use of devices already owned by the participants. Question groups are the only instance were the design intended survey participant to use their own device. The current implementation solely focuses on a shared device scenario. In particular recurring question groups are implemented in the model but cannot be created/deployed with the current implementation of the web interface and the android application.

Even with the aforementioned addition of recurring question groups, the model does not provide anything new or ground breaking. The key feature of the model is that it allows for more than mere question/answer pairs. External code can be loaded into the application at run-time. These code modules are referred to as survey experiments. By supporting survey experiments, the system can go far beyond the capabilities of conventional pen and paper or even online forms. Survey experiments allow researchers to collect more than just textual information. Providing a standardised way of incorporating survey experiments into a survey, has the potential to drastically increase the amount of information that can be gathered with a single survey. Combining survey experiments with the mobility the survey framework offers, results in more data, which inevitably results in stronger evidence for or against a researcher's hypothesis.

Since the experiment's results depend on the nature of the experiment, a single, general question type is required. At its core, every survey experiment can be summarised with the question or a description of the experiment and its result, generated by the participant. Since any result set can be expressed in textual form we define survey experiment questions as questions returning some form of textual response. Whether the response contains timestamps, human readable text or binary data is of no relevance to the underlying model. As the support for different types of experiments is platform specific, details on this topic are discussed in the section Mobile App.

4.2 System components

Figure 3 shows the system and its components.

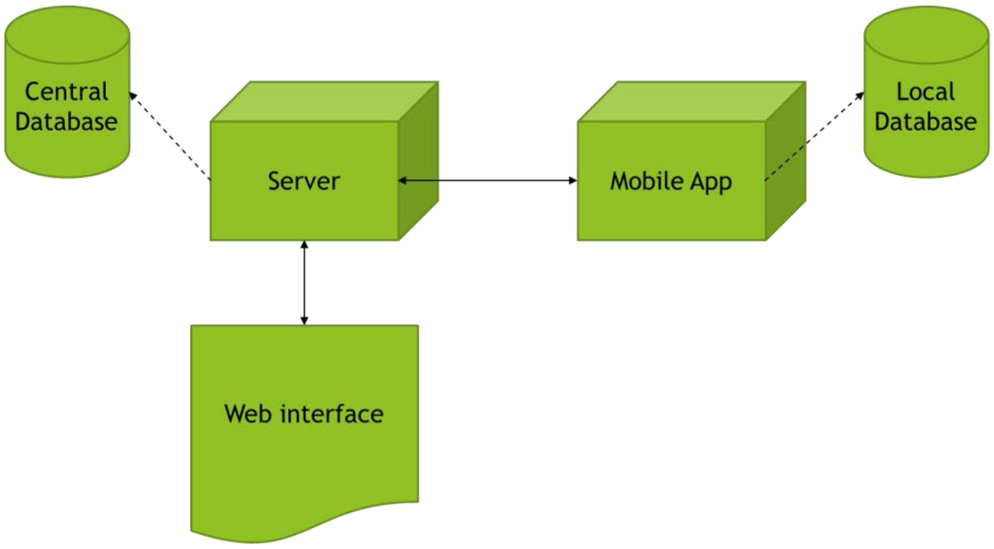
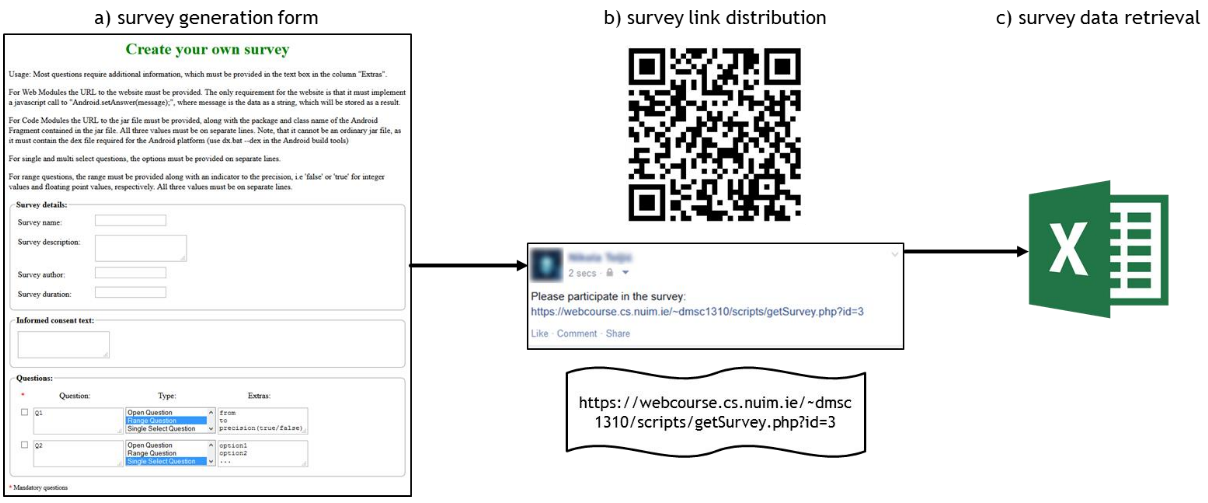


Figure 3: System components: The mobile application displays surveys stored in its database. A separate table stores the survey results. The server uses a web interface for survey generation and stores them and subsequently submitted survey results in a central database

The two main components are the server and the mobile application as they are the core of the back and frontend, respectively. While the researcher interacts with the web interface to create the survey and retrieve the gathered data, the survey participants' main interaction occurs with the application on the mobile device. These two components are now analysed in more detail.

4.2.1 Server

The main purpose of the server component of the system is to provide a central location where surveys and survey results are stored for easy access for participants and for analysis by the researchers, respectively. Additionally, the server hosts the web interface to create surveys and to access the results. The data is stored in a PostgreSQL database. The survey generation is achieved by combining HTML, CSS and JavaScript to create a JSON encoded representation of the survey, which is inserted into the database using PHP. The latter is also utilised to retrieve survey results from the server's database as well as retrieving the survey itself. Figure 4 shows the survey workflow from a researcher's perspective.



Create your own survey

Usage: Most questions require additional information, which must be provided in the text box in the column "Extras".

For Web Modules the URL to the website must be provided. The only requirement for the website is that it must implement a javascript call to "Android.setAnswer(message);", where message is the data as a string, which will be stored as a result.

For Code Modules the URL to the jar file must be provided, along with the package and class name of the Android Fragment contained in the jar file. All three values must be on separate lines. Note, that it cannot be an ordinary jar file, as it must contain the dex file required for the Android platform (use dx.bat --dex in the Android build tools)

For single and multi select questions, the options must be provided on separate lines.

For range questions, the range must be provided along with an indicator to the precision, i.e 'false' or 'true' for integer values and floating point values, respectively. All three values must be on separate lines.

Survey details:

Survey name:

Survey description:

Survey author:

Survey duration:

Informed consent text:

Questions:

*	Question:	Type:	Extras:

☐	Q1	Open Question / [✓ Range Question] / Single Select Question	from
			to
			precision(true/false)

☐	Q2	Open Question / Range Question / [✓ Single Select Question]	option1 option2 ...

* Mandatory questions

2 secs

Please participate in the survey:
https://webcourse.cs.nuim.ie/~dmsc1310/scripts/getSurvey.php?id=3

Like · Comment · Share

https://webcourse.cs.nuim.ie/~dmsc1310/scripts/getSurvey.php?id=3

Figure 4: The researcher's workflow. Firstly, the survey is created. Using various channels the link to the survey can be distributed. Finally, the survey results can be obtained as a spreadsheet.

The survey creation feature is the only one with a graphical user interface, and uses an HTML form to gather the researchers input. JavaScript, and the jQuery framework in particular, allow the HTML form to extend dynamically, allowing the researcher to incorporate any number of survey questions. Once all data is entered and the researcher submits data from the form, a JSON representation of the survey is generated. Subsequently, the survey is sent via a POST request to the server where a PHP script opens a connection to the PostgreSQL database, inserting the survey and returning a link, which is then displayed to the researcher. The link itself points to another PHP script which retrieves the survey based on its ID. The database contains a table *surveys*, with the survey ID as the primary key and the JSON string. The whole process is summarised in Figure 5 below.

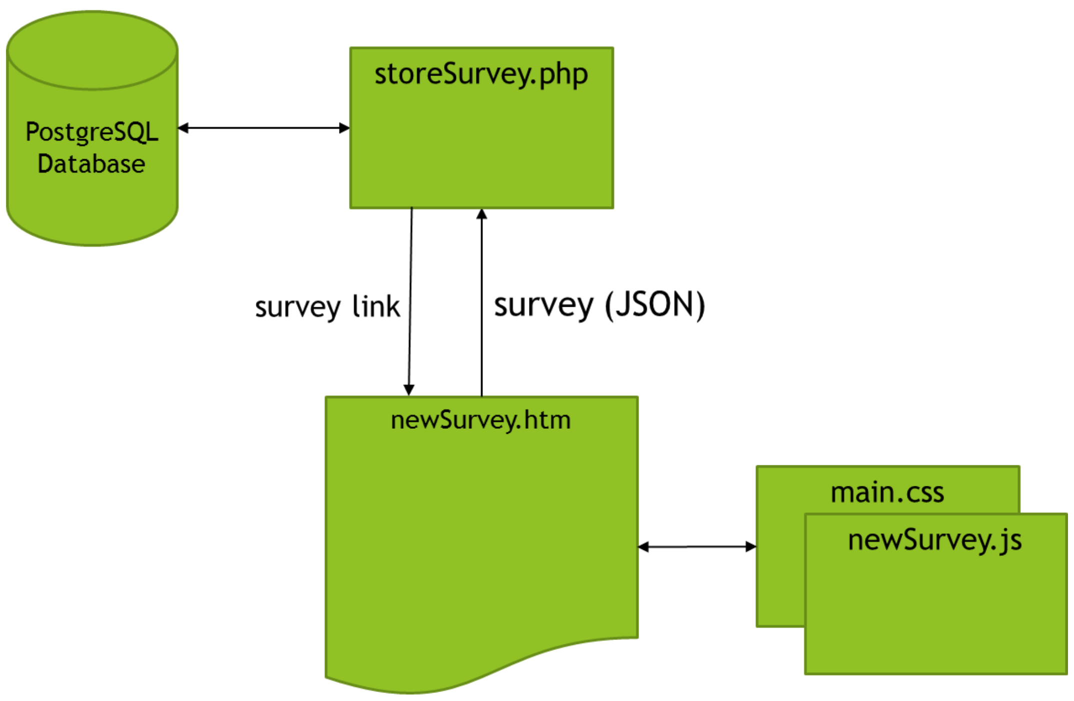


Figure 5: Server components required to create a new survey.

In order to process and retrieve the survey results, several other PHP scripts are used. Firstly, one script accepts survey results as JSON strings, parses it and inserts the data into the database. To accommodate any question type and experiment data the database table, *surveyresults*, is structured as seen in Table 5.

Table 5: The attributes for the surveyresults table. The first three attributes together form the primary key in this table.

surveyID	userID	questionID	answer

The first three elements uniquely identify each tuple. The *surveyID* contains the unique identifier for the survey that is generated automatically when a researchers creates a survey. The *userID* is used to identify individual survey responses. Since a single person could submit results multiple times and a single device can be used by multiple people, the *userID* is simply a timestamp of when the participant started answering the survey questions. This could cause conflicts when a survey is rolled out to many devices. However, the main use case the system is implemented for is one where one device is used by all the participants, the issue could be neglected. It would be quite simple to include a stronger identifier by including an additional attribute storing the unique ID of a device. The *questionID* provides a means of identifying which question this particular answer is associated with. The answer *attribute* stores either the textual response of the participant to the question or a JSON string if the answer structure is more complex. This can be used to store a series of events for a particular experiment as a single answer. Submitting this data in the JSON format is not enforced, but rather recommended as future features of the system could include the generation of a more sophisticated output than a simple csv file with four columns corresponding to the four attributes stored in the database.

This brings us to the other PHP script which generates the aggregated output of all survey responses. It connects to the database, queries it for the tuples matching the specified survey ID and uses PHPs built-in method to return a CSV file.

4.2.2 Mobile App

The core component from a survey participant's point of view is the mobile application. It is how they participate in the survey and can be the only point of interaction, since supervision by the researcher is not necessary; although this depends on the type of survey and on the specific experiments which are being conducted. This section will give a general overview of the functionality offered by the Android application. Technical details will be discussed in Chapter 5.

When the survey application is first installed on a device it does not contain any surveys. Participants can access survey by downloading them using the Android application. Using standard mechanisms provided by the Android platform, the Android application was configured to intercept links matching a specific URL pattern globally on the device. Whenever a user attempts to navigate to a link matching the pattern a dialog is shown. The dialog lets them chose whether to open the link in the survey application or in a web browser. Two patterns are specified, one for surveys retrieved from the database accessible via

https://webcourse.cs.nuim.ie/~dmsc1310/scripts/getSurvey.PHP?id=xx,

providing the ID of the survey. The second pattern intercepts links to surveys stored in the *surveys* directory on the server and file names ending with '.json'. Once the survey data is downloaded, it is then stored in a local database on the Android device to avoid unnecessary network connections, thereby keeping the applications energy consumption as small as possible.

Since the survey can contain experiments which require their own code and/or data to be executed properly the application identifies all URLs directly specified in the survey, downloads the content and stores it locally. This requires the devices to be connected only initially. For basic functionality of the application a data connection is not required after this point, allowing surveys to be conducted outside the reach of wireless networks. It is important to note however, that third party modules can require a network connection to be present during the experiment. This is not checked by the application and therefore it is up to the developer of the module to ensure proper error handling and fall back mechanisms are included. An internet connection is of course required to upload the survey results. The upload is triggered manually by the researcher.

4.2.2.1 Question groups

Since the idea of question grouping was already discussed previously, this section will focus on the difference between the system model and the actual capabilities, in regard to question groups, of the mobile application. The idea behind the model is that each survey participant can be presented with a set of one time questions and sets of recurring questions. Due to the fact that user re-identification was not implemented, the only scenario were this capabilities could be put to use is one where every survey participants uses an individual device. However, the survey conducted for this project used only one device for all participants. Therefore the implementation does allow one-time question groups to be answered multiple times.

Moreover, while recurring question groups can be displayed they do not behave any differently from one-time question groups. Also the mechanisms for active recurring question groups are not in place. Since the application does not support these features, neither does the survey generation tool. This means that, while the design model and implementation do not match perfectly, the server side and the mobile application do support the same functionality. Chapter 8 explains how this issue can be addressed.

4.2.2.2 Built-in question modules

For every question type discussed earlier a custom question module was created for the survey application. This allowed for a quick and intuitive interaction with the application. Open ended questions use a simple text box for user input, range questions display a slider bar and select questions present the participant with the options in form of a list. Depending on the type of question chosen the number of items that can be selected varies.

4.2.2.3 External modules

The application supports three different types of third party modules. Web modules simply display the URL that is provided by the researcher. While the content of the URL is downloaded and stored on the device, the same is not true for content referenced within the resources, e.g. images within HTML pages still require an active internet connection to be displayed. Navigation through links is disabled to keep the interaction simple for the user. For the same reason there is no address bar where the user could enter a URL manually as a web module is not a full-fledged web browser but rather a special purpose component which can display a single online resource. Interaction between the web module and the survey application is achieved using a JavaScript interface that the web module must implement.

Android modules must be implemented by sub classing the Fragment class which are provided by the Android framework. Fragments are an integral system component of every modern Android application and have a special lifecycle that is managed by the operating system. The implications of this for module integration are discussed in Chapter 5. In addition to being a subclass of Fragment a module must also implement the `FragmentModule` (highlighted text refers to class names within the application) interface provided by the survey framework. The methods specified by this interface provide hooks to event within the survey fragment. Further details are discussed later.

Unity modules provide researchers with a powerful gaming engine which they can leverage when designing their experiments. Similar to Fragments on the Android platform, Unity applications have modular components called Scenes. While the survey platform does not impose any special restriction on the design of Unity modules, it was only tested with modules containing one Scene. Since the focus of the project was on creating a survey platform, no significant amount of resources was allocated to investigate the limits of the Unity engine for the Android platform when used within the framework. Communication between Unity components and the survey application occurs via a predefined Java interface which can be called from within the Unity component. The exact interaction is discussed later.

Once the participant has answered all the mandatory questions they can submit the results. To facilitate offline use of the application the results are stored locally on the device using a SQLite database. The structure of the table is almost identical to the table *surveyresults* (names in italics refer to table names in the database) on the server side. The only difference is the addition of the attribute *uploaded*, which stores a flag indicating whether this particular response has already been successfully uploaded to the server. For each survey on the device the researcher can choose to upload all remaining survey responses when they see fit, i.e. when a data connection is available. The data is converted into JSON and processed by a PHP script, which inserts the data into the postgreSQL database on the server.

4.2.3 Limitations

The system implemented is stable and ready to use in-house to showcase its capabilities. It is not, however, ready for a public release. This section elaborates on the missing functionality that is not required to demonstrate the underlying concepts of the survey framework but would be critical components in a publically released version.

While it is perfectly reasonable to allow public (read) access to the surveys stored on the server it would be highly irresponsible to allow the same level of access to survey results, mainly because there is no control over what questions will be asked by researchers. In order to protect this potentially sensitive information access control needs to be put in place to ensure that only authorised users of the system can access the data. From an engineering perspective this could be easily achieved by requiring researchers to register before they can use the system. They would then have personal login credentials that would make it straightforward to manage access control.

Data must not only be properly secured on the server side. Despite the sandboxing and access control mechanism in place on the Android platform, there are ways to access the private data of any application, including survey results temporarily stored on the device. Storing the data in an encrypted format would prevent any unauthorised access to survey results stored on mobile devices. This is not only a sensible thing to do but also required by the Data Protection Act [34].

The survey generation form currently accepts any input to the form fields. Even though the system is safe from SQL injections, incorrect input can cause application crashes when the users attempts to participate in a survey. Therefore input validation is a vital component that would need to be implemented for a publically accessible system. This issue only pertains to the survey generation form and not the Android application. Survey participants’ input in the mobile application is limited to valid inputs through the use of slider bars and drop down menus.

5 Android survey framework application

This chapter provides an in depth discussion of the technical aspects of the system. Brief descriptions of the platforms, frameworks and API used are also provided. Some of the details covered here however, might require a deeper understanding than can be covered in this document as they are fairly technical. The interested reader is invited to refer to Chapter 2 for material on e.g. the Android operating system and framework. Many implementation and design decision were based on how the operating system handles the life-cycle of application components.

5.1 Model

This section describes the classes provided by the operating system and the classes created specifically for the survey framework data model. Any reference to class names or object methods are highlighted, to help the reader distinguish between for example the noun "question" and the java class `Question`. At the core of the framework is the Java class `Survey`. Apart from storing the survey's identifier, name, description and the authoring institution, it also stores all the questions and experiments. Because of the question groups explained in the chapter System overview, questions are not stored in a simple data structure like an array or list. Instead, each `Survey` instance holds a reference to two `QuestionGroup` instances, an `OneTimeQuestionGroup` and a `RecurringQuestionGroup` (*RQG*). A `QuestionGroup` object holds a list of questions and a list of sub groups. Sub groups can only be of the same type as the parent `QuestionGroup`. This is achieved through the use of Java Generics. The signature of the `QuestionGroup` class looks as follows:

`public abstract class QuestionGroup<T extends QuestionGroup<?>>`,

with the method signature for adding groups being:

`public void addSubGroup(T group)`.

Now, when the *RQG* is defined as

`public class RecurringQuestionGroup extends QuestionGroup<RecurringQuestionGroup>`, its `addSubGroup()` method only accepts `RecurringQuestionGroup` as a parameter. The same applies for the return type of the method `getSubGroups()`.

While sub groups can be added to all `QuestionGroup` instances, adding groups to an `OneTimeQuestionGroup` object does not change its behaviour. This is not the case with *RQG*, where subgroups can represent independent sets of questions, depending on their state. A *RQG* is defined to be independent if it contains questions that represent a self-contained set of questions. This is indicated by the existence of a non-null `Timing` member. The class `Timing` holds information on when questions within a group should be answered.
Independent RQGs have an additional flag indicating whether they are an active RQG. This means that the participant should be actively notified when it is time to answer the questions within this RQG (according to the `Timing` member). The UML class diagram shown in Figure 6 gives an overview of the relation between `Survey`, `QuestionGroup`, `Timing` and `Question`. For the sake of clarity only the class names are shown. For a more detailed figure refer to appendix A.

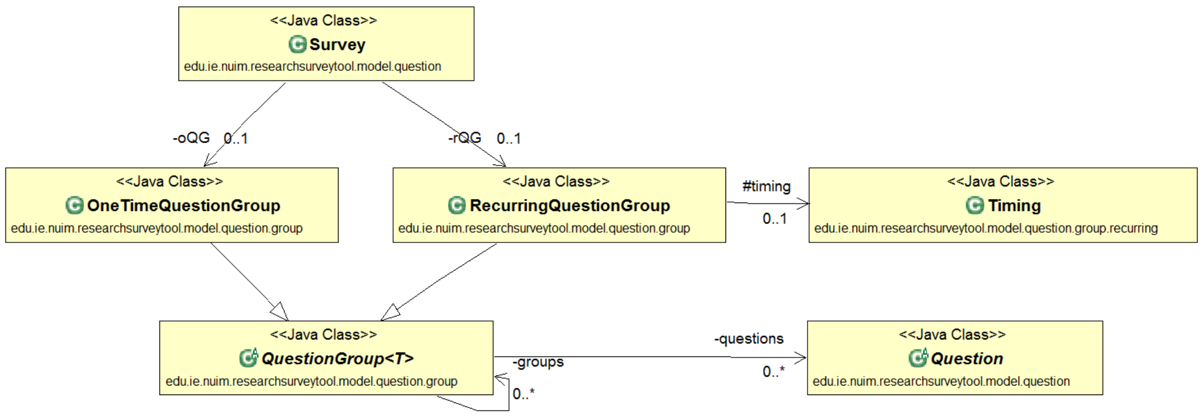

<<Java Class>>
Survey
edu.ie.nuim.researchsurveytool.model.question
-oQG 0..1
-rQG 0..1
<<Java Class>>
OneTimeQuestionGroup
edu.ie.nuim.researchsurveytool.model.question.group
<<Java Class>>
RecurringQuestionGroup
edu.ie.nuim.researchsurveytool.model.question.group
#timing
0..1
<<Java Class>>
Timing
edu.ie.nuim.researchsurveytool.model.question.group.recurring
<<Java Class>>
QuestionGroup<T>
edu.ie.nuim.researchsurveytool.model.question.group
-groups
0..*
-questions
0..*
<<Java Class>>
Question
edu.ie.nuim.researchsurveytool.model.question

Figure 6: Class diagram showing the relation between the core model classes. A survey object contains two different types of question groups which in turn can have their own, sub question groups and/or question.

5.2 Activity

This section covers the Android specific implementation details of the system. First, the key Android components provided by the software development kit (SDK) are introduced. Next, the different possibilities of employ the SDK components to implement a dynamic surveying tool are compared and their advantages and disadvantages analysed. Finally, the concrete implementation of the survey framework is detailed.

Unlike regular desktop java applications that can use Swing, AWT, SWT, JavaFX or any of the many other GUI frameworks for their graphical interface, Android comes with its own GUI API. It is tightly coupled to the core components `Activity` and `Fragment`. An `Activity` represents a single screen within an Android application. They are the backbone of every Android application that has a GUI. Their life-cycle is handled by the system and it is up to the developer to implement the hooks into the life-cycle method calls appropriately, to ensure that the `Activity` behaves properly. Figure 7 was taken from the android developer website and shows the life-cycle stages of an `Activity` including the most common transitions.

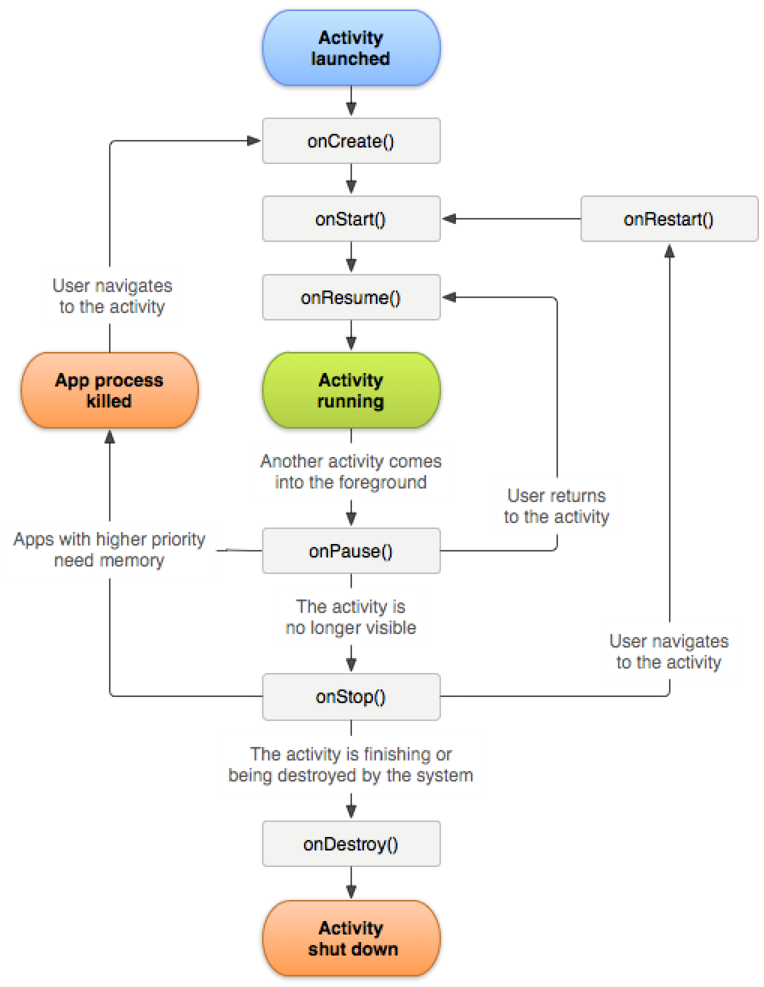

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 7: Activity life-cycle diagram as used in the official android developer documentation. The main methods usually overridden in Activity subclasses are onCreate(), onResume(), onPause() and onStop().[1]

Figure 7 illustrates how relatively complex an `Activity` is. Only a few properties of the `Activity` class are discussed here, as they are the most relevant to the design decision of the survey system. In particular it is of importance to understand what `Fragment` and `Activity` components have in common and were their key differences are. Every subclass of an `Activity` that is used in an application must be declared in the application's manifest. The manifest is an xml file that stores public properties of the application, such as the package name, application name, required security permissions and applications components, including `Activity` components. In addition to the above transitions, an `Activity` can be re-created by the system under

[1] Figure obtained from: http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle

certain circumstances. `Activity` components in their paused state for example will be killed by the system if the current foreground `Activity` requires more resources. If the user navigates back to a killed `Activity` it is re-created by the system. To ensure proper re-creation the state of the `Activity` (and its associated data) must be preserved. Another case were an `Activity` is re-created is when the device configuration changes. Such changes include, but are not limited to, screen orientation changes, system language changes and default font size changes. In such cases the life-cycle methods may be called in quick succession. Nonetheless, the state of the `Activity` that is about to be re-created must still be persisted. This re-creation process, while necessary to support all the different types of device configurations, can cause problems when `Activity` components outsource resource intensive creation tasks to other threads. If an application for example, loads and displays a web page and the user rotates the devices from landscape to portrait mode, the web page is reloaded if the state changes are not handled properly, i.e. the web page is persisted temporarily to avoid unnecessary network traffic. If the timing is particularly bad, the background thread loading the web page (it is common practice to move tasks that might block the UI thread to a worker thread) returns the data to the `Activity` just after it was destroyed, resulting in a `NullPointerException`. Properly handling such situations requires careful consideration from the developer, this is discussed further in section 5.5.

5.3 Fragment

`Fragment` is a complementary component to `Activity` that can also be used to define parts of the application layout. However, `Fragment` components cannot be used as standalone components, rather they need an `Activity` to host them and can either define the layout of an entire screen or just part of it. They too have a life-cycle that is handled by the system and is often tied to the life-cycle of the `Activity` hosting the `Fragment`. Most of the life-cycle methods shown in Figure 7 also apply to the `Fragment` class. On the other hand, `Fragment` components are not declared in the application's manifest. In addition, they can host other `Fragment` components. Such nesting is not possible with `Activity` components. Furthermore, in contrast to the

`Activity` class, instances of `Fragment` can be retained across configuration changes, preserving all its contained objects and its data, making the handling of concurrent tasks easier. Another key difference between the two is that moving from one `Activity` to another requires the use of the Android intent system, whereas `Fragment` transitions are managed by the `FragmentManager` of the hosting `Activity`. `Intent` objects can pass on data but the encapsulating object must either implement Java's `Serializable` interface or Android's equivalent `Parcable` interface. On the other hand, since changes of `Fragment` components occur within a single `Activity`, data can be passed back and forth between those components much more easily.

5.4 One Fragment per Question

In general, `Activity` and `Fragment` are the two options to provide a GUI for survey questions within the application. `Fragments` were chosen as they are more lightweight components and allow nesting. They can also be retained across configuration changes. Furthermore, communication between `Fragment` instances within the same `Activity` object can be done with little overhead. Another reason to favour `Fragments` over `Activities`, is the system's use of the `ViewPager` component provided by the Android SDK. A `ViewPager` is a UI component which provides lateral navigation between its sub UI components, in this case `Fragments`. Once the `ViewPager` is set up properly, it also handles all the `Fragment` transitions that one would have to deal with otherwise. It also allows for an arbitrary number of `Fragments` and therefore an arbitrary number of questions to be included.

5.5 Application structure

This section shows how the classes from the survey model interact with their Android counterparts. Figure 8 illustrates the relations between the individual components in a `QuesitonGroupActivity`, which is responsible for displaying all questions within its associated question group.

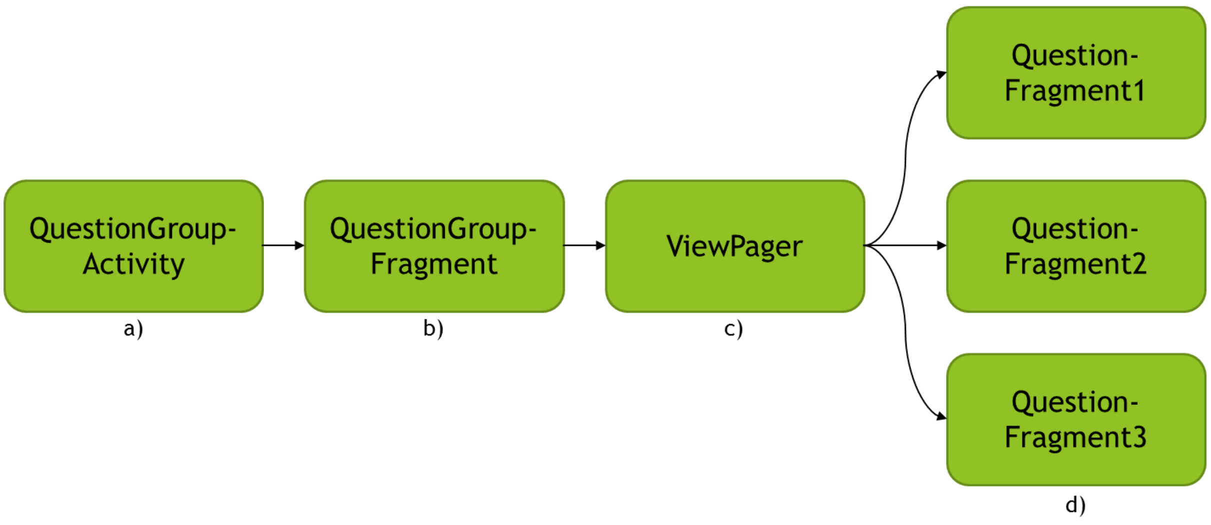


Figure 8: The component structure within a QuestionGroupActivity. The QuestionGroupFragment allows its containing data to be retained, which is much easier done with Fragments compared to Activities. The ViewPager hosted in the QuestionGroupFragment itself hosts the actual Fragments representing survey questions. Each question is stored in its own instance of class Question and is displayed in an instance of class Fragment.

The main component within a `QuestionGroupActivity` is its `QuestionGroupFragment`. While this setup causes some overhead since the `ViewPager` could be hosted by the Activity directly, it proves highly efficient when considering the fact that the Activity is destroyed and recreated when a configuration change occurs. This would mean that at the very least the `ViewPager` component would need to be recreated as well. To ensure a consistent user experience, the state of the `ViewPager` would need to be stored temporarily so it can be restored in the new instance. This means that all `Fragment` components and their position within the `ViewPager` must be saved. By placing the `ViewPager` object into the `QuestionGroupFragment` retaining it as well as all other Fragment that represent questions, very little overhead is required to properly handle configuration changes. In general, not all `Fragments` can be retained, however, as stated in the documentation for `setRetainInstance:`

“THIS [METHOD] CAN ONLY BE USED WITH FRAGMENTS NOT IN THE BACK STACK.”

The back stack, in this case, refers to `Fragments` the user navigated away from within the same Activity, provided that `addToBackStack()` was called for them. This would be an issue if, instead of using the `ViewPager`, navigation between `Fragments` would have been implemented manually, since the easiest way to maintain a history of displayed `Fragments` is to add them to the back stack. The `ViewPager` however, manages `Fragments` automatically, therefore providing the desired features without the additional coding overhead that a custom navigation component would bring. This justifies the additional effort put into setting up the `ViewPager`.

The actual `Fragments` being displayed by the `ViewPager` have a one-to-one mapping with the question types introduced in the chapter System overview. Figure 9 shows this mapping. Additional classes are shown that have a different mapping. They are explained in the next section.

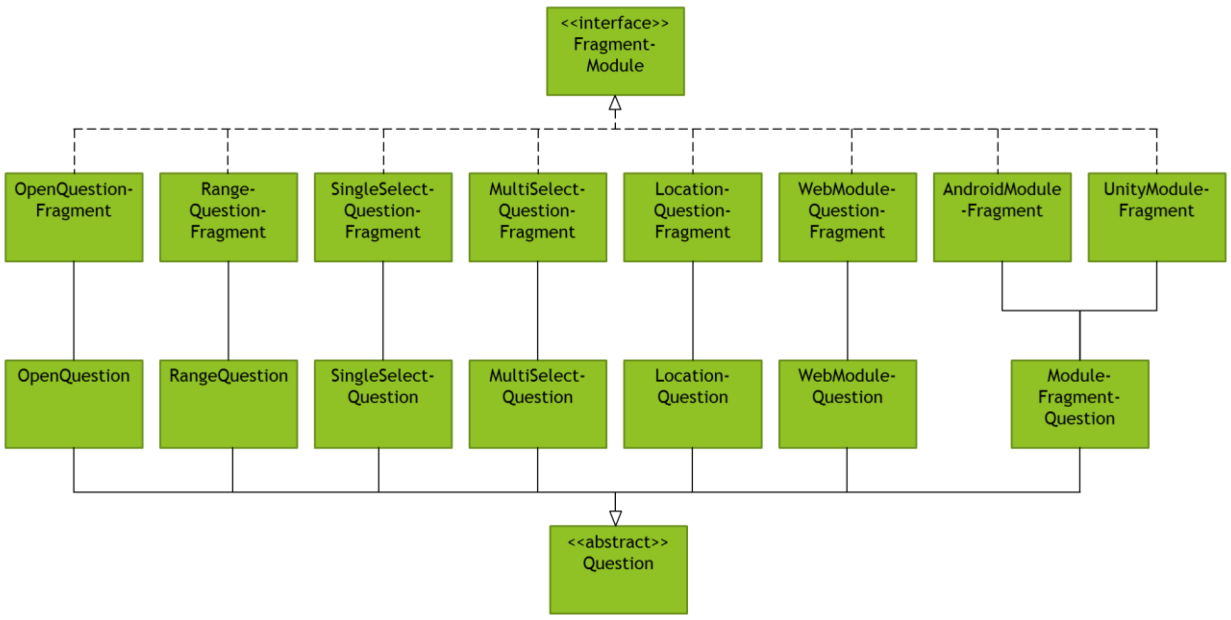


Figure 9: Class diagram showing the mapping between QuestionFragments and their corresponding Question subclasses.

All `QuestionFragments` are subclasses of the `Fragment` class provided by the Android system and there is no actual class `QuestionFragment`. Instead the `FragmentModule` interface is defined to ensure that functionality common for all `QuestionFragments` is implemented. Because the methods defined in the `FragmentModule` interface have almost identical implementations across all `QuestionFragments` it might seem that it would be a better approach to create an abstract class which implements the common functionality and allows for methods to be overridden were necessary. However, while most of the classes are direct subclasses of Fragment some are indirect subclasses and have `ListFragment`, another system provided class, as their direct superclass. Instead of creating a `QuestionFragment` and a `QuestionListFragment` the interface approach was chosen. The side benefit of the interface approach is that it offers greater flexibility for extensions of the system as it allows e.g. an `Activity` to implement the `FragmentModule` as well, whereas otherwise the system would be strictly limited to the use of `Fragments`.

Using the approach outlined in this section a mapping is created between `Fragment` classes implementing the `FragmentModule` interface and their corresponding `Question` classes. Additionally `Fragment` retention and the back stack, two concepts from the Android framework, were discussed. The `ViewPager` component was introduced in this chapter to show that it is a vital part of the structure of the Android application. However, the next section explains the inner working of the `ViewPager` component in more detail.

5.5.1 ViewPager & FragmentModule

This section gives a detailed description of how the `ViewPager` component works and how it interacts with `Fragment` instances that implement the `FragmentModule` interface. Its main purpose is to show why the chosen application structure is necessary and skipping it will not negatively affect the understanding of the remaining sections in this chapter.

In order to understand the necessity for the methods defined in the `FragmentModule` interface it is important to realise the exact workings of the `ViewPager` class. The `ViewPager` instance is backed by subclass of `FragmentStatePagerAdapter` (`Adapter`), which is responsible for providing the `Fragments` to be displayed to the `ViewPager`. Only two methods need to be overridden in the subclass, namely `public Fragment getItem(int pos) and public int getCount()`. The latter returns the total number of elements for this `Adapter` and the former provides the `ViewPager` with the `Fragment` belonging to the specified position. The `ViewPager` itself requires the addition of a `PageChangeListener`, which receives callbacks when the user scrolls through the `ViewPager` and when a page (i.e. `Fragment`) is selected. While the former is irrelevant in this context, the latter callback is the key to achieving the desired behaviour of the `ViewPager`. The `ViewPager` class has no default mechanism to prevent the user from advancing through its pages. Some questions however, might be mandatory and it should therefore not be possible to simply skip them. The functionality is achieved by overriding two methods of the `ViewPager` that handle touch input, `onInterceptTouchEvent()` and `onTouchEvent()`. Both methods deal with `MotionEvents` rather than clicks, therefore it is safe to manipulate them without risking adverse effects when the user touches the screen with the intent of clicking a child element, e.g. a button within a `Fragment`. In order to intercept the `MotionEvents` received by these methods it is enough to not call their respective superclass methods. To determine whether the user can scroll through the pages, every time a `MotionEvent` is received, the currently displayed `Fragment`, which is being kept track off by the `PageChangeListener`, is queried by calling its `canSkip()` method as defined in the `FragmentModule` interface. Depending on the returned value the superclass method of the method that received the event is called or not.

The `FragmentModule` interface is also needed for another reason. In order to ensure a smooth scrolling experience for the user the `ViewPager` loads `Fragments` before they are needed. By default it keeps references to three `Fragments`, the currently displayed one and the two `Fragments` immediately to its right and left. This means that the `Fragment` code will be executed before the `Fragment` itself is visible to the user. While most `QuestionFragments` are not affected by this, some are. In order to delay code executing that requires the user to see what is happening the `PageChangeListener’s onPageSelected()` method is used. Whenever a page (i.e. `Fragment`) is selected its `onShow()` method is called. The page that was removed from the screen by this action receives a call to `onHide()`. The `Fragments` affected by this issue are covered in the next section.

Once the `ViewPager` is setup correctly it allows lateral navigation between questions by swiping the screen. In addition it manages all the `Fragment` instances that are contained within the question group which the `ViewPager` is displaying.

5.5.2 Experiments & Third party modules

The key piece of functionality that differentiates this survey framework from the many survey applications available on the market[2], is that it allows researchers to include experiments into their surveys. Not only can they add online resources such as HTML pages but also experiments specifically written for the Android platform, leveraging a lot of the capabilities offered by the mobile device. Furthermore, the framework supports third party code written for the Unity platform, a game engine that allows for realistic physics simulations. All three approaches allow the reuse of existing code and only require the implementation of the appropriate interface to allow communication between the third party components and the survey application itself. This section shows how third party modules are integrated with the existing components of the application, how each of the different module types interacts with the framework and most importantly, how, for each type, the code is added to the application dynamically at runtime.

As seen in Figure 9 additional `Question` subclasses are added to accommodate for the additional information that needs to be stored for those types of questions. Instead of the typical one-to-one mapping the `ModuleFragmentQuestion` class is used for both, Android and Unity modules.

5.5.2.1 Web modules

`WebModuleQuestions` have a quite straightforward implementation since the Android system provides everything that is needed to download and display online content. The `WebModuleQuestionFragment` contains a `WebView` component, which is used to display

[2] https://play.google.com/store/apps/details?id=net.pocketsurvey.android
https://play.google.com/store/apps/details?id=com.surveypocket
https://play.google.com/store/apps/details?id=dooblo.surveytogo
https://play.google.com/store/apps/details?id=appdictive.instasurvey
https://play.google.com/store/apps/details?id=com.shikshainfotech.customersurvey
https://play.google.com/store/apps/details?id=com.qscript.demo
https://play.google.com/store/apps/details?id=com.tsurveys.application
https://play.google.com/store/apps/details?id=de.cluetec.mQuestSurvey

the online content (e.g. HTML page). Instead of fetching the data every time the user answers the questions, it is stored locally on the device. This allows the use of the application even when no internet connection is available. The current implementation only stores the URL specified in the `WebModuleQuestion`. Referenced content within the specified resource are not preloaded. This means that images in HTML pages still require an active internet connection to be displayed. Interaction between the online resource and the `WebModuleQuestionFragment` is facilitated by combining JavaScript and the `FragmentModule` interface implemented by the `WebModuleQuestionFragment`. For the `WebView` component to react to JavaScript method calls, JavaScript must first be enabled for the `WebView` itself, as it is disabled by default for security reasons. Furthermore an object implementing the `JavaScriptInterface` interface must be added to the `WebView` by calling its `addJavascriptInterface()` method. Once the `WebView` is set up, Android method within the scope of the `WebView` can be called in JavaScript by using the prefix 'Android.'. Only methods with the `@JavaScriptInterface` annotation can be executed using this approach. As the `WebModuleQuestionFragment` implements both, the `FragmentModule` and `JavaScriptInterface` interfaces, the answer to the `WebModuleQuestion` can be set by calling `Android.setAnswer()` from the JavaScript script and passing the answer string as a parameter.

As all questions have a corresponding `Fragment` class which implements the `FragmentModule` interface it is fairly easy to see that in order to allow experiments, all that is really necessary is a way to provide room for custom questions. Therefore, all that is needed to create an experiment is to create a subclass of `Fragment`, a requirement that is in place because questions are displayed in a `ViewPager` for `Fragments`, and for it to implement the `FragmentModule` interface. Conceptually and actually, this is all a researcher needs to know in order to provide their own custom question types or experiments. Integrating those experiments (read: external code) is one of the key features of the survey application. To understand the complexity of, not only loading external code, but also managing the loaded `Fragment's` life-cycle appropriately, the principles necessary for this understanding are laid out in the following paragraphs.

5.5.2.2 Dynamic class loading

Most android developers write their application code in Java, although using C with Android's NDK (Native Development Kit) is also possible. The use of Java would normally indicate the presence of a JVM (Java Virtual Machine) on the device it is running on. Instead, rather than using the compiled Java byte code directly, it must be recompiled into Dalvik byte code since Android applications run on a DVM (Dalvik Virtual Machine). While Java byte code is stored as *.class* files, Dalvik byte code is stored in *.dex* files. Additionally, all .class files created for any one particular Android application are recompiled into one single *classes.dex* file. For regular Android applications this file is packaged together with pre- and uncompiled resources (strings, bitmaps, binary data, etc.) and a compiled version of the manifest file into an *.apk* file. These are the basic steps needed to create a runnable Android application from Java source code. Other steps, such as aligning the package and signing it are an essential part of building an app before publishing it; however those steps are ignored for the sake of brevity and clarity.

One of the main strengths of Java is the abundance of APIs available for it. One such example is the `ClassLoader` API. As the name suggests and the Java documentation confirms:

"A class loader is an object that is responsible for loading classes."[3]

As further explained by the documentation, typically a *.class* file is read from the file system to make the class definition of the desired class available. While `ClassLoader` is an abstract class its subclasses `SecureClassLoader` and `URLClassLoader` are concrete classes that can be used to load classes. The latter simply takes a `URL` of the class to be loaded, providing developers with an intuitive way of extending their application's code dynamically. While those classes are also available within the Android framework, they are not equipped to handle *.dex* files since *.class* files are effectively useless on an Android powered device. Fortunately, the Android

[3] http://docs.oracle.com/javase/7/docs/api/java/lang/ClassLoader.html

framework also provides the classes `BaseDexClassLoader`, `DexClassLoader` and `PathClassLoader`. These classes are specifically designed to load classes from *.dex* files. While the first one serves as a base class, containing common functionality across its subclasses, the other two allow classes to be loaded from *.jar* or *.apk* files, and from files contained in a directory, respectively. Since `DexClassLoader` cannot load classes from *.jar* or *.apk* files directly, it requires access to an application-private, writeable directory to store the unpacked *.dex* files from which to load classes, as explained in the documentation.[4] `DexClassLoader` was found to be the most suitable `ClassLoader` for our survey application. The following code snippet shows the usage of the `DexClassLoader` within the framework.

```
File dex = ctxt.getDir("survey_"+question.getSurvey(), Context.MODE_PRIVATE);
DexClassLoader classLoader = new DexClassLoader(path, dex.getAbsolutePath(),null,
                              ctxt.getClassLoader());
String className = question.getPackageName() + "." + question.getFragmentName();
Class<?> c = classLoader.loadClass(className);
Fragment f = (Fragment)c.newInstance();
```

Listing 1: Sample code loading an external class from a .dex file

The `DexClassLoader` constructor takes four parameters, with the first two specifying the path to the source *.jar* or *.apk* file and the path to the location where the unpacked *.dex* files are to be stored, respectively. Both parameters are determined by the `Question` object holding information about the experiment. The third parameter, which in our case is `null`, serves to add paths to libraries. Since `DexClassLoader` is a subclass of `ClassLoader` it also follows the same hierarchical where it delegates any `loadClass()` calls to its super class and only loads the class itself it its parent is unable to do so. Therefore, the parent needs to be provided with the `DexClassLoader` constructor (last parameter). In this case it is the `ClassLoader` retrieved from the `Context` object associated with the `Activity` that holds the component which initiated the class loading. Once the `DexClassLoader` is constructed its `loadClass()`

[4] http://developer.android.com/reference/dalvik/system/DexClassLoader.html

method is called with the fully qualified name (package + class name) of the Fragment to be loaded, as a parameter.

This concludes this section which showed the basic idea of how the `ClassLoader` API can be used to create object instances of dynamically loaded classes in Android.

5.5.2.3 Android modules

Creating an instance of the externally supplied `Fragment` is the first big step; the second one is adding it as a page in the `ViewPager` and ensuring that it is maintained properly throughout the containing `Activity's` life-cycle. Including the `Fragment` into the `ViewPager` can be done in one of three ways. The most straightforward approach is to simply add the Fragment to the `ViewPager's Adapter` and to call its `notifyDataSetChanged()` method, which will trigger the `ViewPager` to reload the `Fragment` for its current position. If the Fragment is added before the currently displayed position the `ViewPager` will now display the previous question since it is not at the position the previously displayed `Fragment` was at. However, if the `Fragment` is added after the currently displayed position, the only change that is apparent to the user is that the indicator, showing how many questions are in the current `QuestionGroup`, will be adjusted. Both scenarios are not ideal, with the former one being very disruptive for the user as it causes a sudden question change that is caused by user interaction. The second possibility is to create placeholder `Fragments` that are replaced when the proper `Fragment` is loaded and instantiated. This approach causes no sudden changes in the UI except when loading the actual `Fragment` that is to replace the placeholder, at which point the screen would change from a loading symbol to the actual experiment UI. From a user's perspective this is certainly the better approach.

Nonetheless, both the aforementioned solutions have a significant drawback, namely that the external `Fragment` is managed directly by the `ViewPager` and its `Adapter`. This in itself might not seem like a particularly bad situation, but it causes major issues when the device undergoes a configuration change after the external `Fragment` has been added. As previously mentioned, when a configuration change occurs all `Activities` are re-created. This includes its contained `Fragments` unless they are retained by a call to `setRetainInstance(true)`. If the external `Fragment` is not retained, the system attempts to create a new instance of it and discards the old one. But as the system class loader has no reference to the class because it was originally instantiated by a custom class loader, the application will crash with a fatal `RuntimeException`. By retaining the `Fragment` the system has no need to attempt recreating a new instance of the external `Fragment` and the issue can therefore be avoided. Despite it being such seemingly easy solution, retaining `Fragments` only alleviates the problem's symptoms without dealing with the problem itself. As mentioned previously, configuration changes are only one possible trigger for `Activity` re-creation. There are also others which will force the `Fragments` to be destroyed as well. While they might not be of particular relevance to the default use case, envisioned for the survey framework, the very fact that it is a framework for others to use implies that one cannot rely that everyone will adhere to the default use case. Therefore the decision was made to not retain externally loaded `Fragments` as the exact workings of the `Fragment` cannot be known. Therefore, a third party module might also rely on the fact that it will not be retained across configuration changes.

The third solution, which is also the chosen implementation for the system, takes advantage of the nesting capabilities of `Fragments`. Instead of replacing the actual module for the placeholder `Fragment`, the module is inserted into the placeholder. This maintains the advantage of the previous solution that the number of questions is known to the participant form the start and it also works around the `Activity` re-creation issue. The placeholder `Fragment` is retained and the actual module kept as a member, effectively retaining it too. In order to avoid putting restriction on `Fragment` retention on external modules, the placeholder checks if the implicitly retained module actually wants to be retained. If not, the placeholder simply discards the old module and executes its own logic which re-creates the module using the `ClassLoader` approach discussed previously. While this solution may seem fairly straightforward it conflicts with the aforementioned restriction that nested `Fragments` cannot be retained. To circumvent this restriction the nested `Fragment` is managed by the `Activity's` `FragmentManager` rather than the placeholder's child `FragmentManager`. While this solved the retention issue, it brings back the problem that the Activity is asked, by the system, to re-create the `Fragment`, resulting in the same

`RuntimeException` as encountered before. This is avoided by storing the retention state of the module separately within the placeholder `Fragment`, which has its own logic for retaining or re-creating the contained module, and calling `setRetainInstance(true)` on the module `Fragment`. This tells the `Activity` to keep the instance, but it might be replaced by the placeholder `Fragment` after the `Activity` is successfully re-created.

This section described how external Android modules are handled by using placeholder components. Rather than replacing the placeholder component, the external module is loaded within the placeholder to ensure that it can be retained across orientation changes. Without the use of placeholders the Android application would simply crash anytime the `QuestionGroupActivity` is recreated by the system.

5.5.2.4 Unity

Apart from just handling simple Android code the framework also allows researchers to include Unity based programs into their surveys. The intention was to provide researchers with a way to deploy experiments with complex physics simulations mimicking some aspect of the real world. The Unity platform was chosen because it offers good integration with the Android platform and it is free to use unlike other solutions such as Monogame for Android. This section gives an introduction into the application development with Unity and elaborates on the communication between the Unity program and the survey framework, how the framework accommodates the external code and what researchers must do to add a unity module to their survey.

Unity is a cross-platform game engine, currently supporting every major desktop and mobile operating system. Developers can use a combination of C#, JavaScript inspired UnityScript and Python inspired Boo for the program logic. The game environment is created within the Unity IDE with support for a wide variety of asset resources. Program logic can be attached to individual virtual objects which are executed by the unity player, controlling the objects behaviour. Similarly to the Android components Activity and Fragment, the system calls predefined methods at certain points in time. Unlike Android though, were these calls manage the life-cycle of components, Unity provides hooks to initialise game objects and their behavioural logic. The main methods are shown in Table 6 with a brief description of their intended use. For most game objects, logic will be executed in one of these methods.

Table 6: Most frequently used life-cycle methods within a Unity application.

Method	Description
Start	Called once, to do any initialisation work.
Update	Called every frame to update the object.
LateUpdate	Called every frame after OnUpdate has been called.
OnCollsionEnter	Called when two `rigidbodies` collide
FixedUpdate	Called at a fixed interval and intended for physics calculations.
OnGUI	Called every frame to draw any GUI components

Where the online resource displayed in the survey framework's `WebModule` worked with Android's `JavaScriptInterface`, communicating between the Unity program and the survey application requires some more work. The `UnityPlayer`, which is the component executing the Unity program, has a custom implementation depending on the platform it is designed for. In the case of the Android platform, the `UnityPlayer` has a reference to the current `Activity` it is executed in. A reference to it within a script (UnityScript) can be obtained with code shown in Listing 2.

```
var className : String = "com.unity3d.player.UnityPlayer";
var activity : String = "currentActivity";
var cPlayer : AndroidJavaClass = new AndroidJavaClass(className);
var oActivity : AndroidJavaObject = cPlayer.GetStatic.<AndroidJavaObject>(activity);
```

Listing 2: Within the Unity script, instances of Android classes can be created and accessed. This mechanism is used to send Unity data to the Android survey framework.

The classes `AndroidJavaObject` and `AndroidJavaClass` are Unity representations of instances of Java objects and classes, respectively. Whether Unity uses the Java Reflection API or a different mechanism was not explored and is not relevant to the functionality of the survey framework.

One the `Activity` instance is obtained any methods contained within it can be called using the `AndroidJavaObject’s` `Call` and `CallStatic` methods. For the survey framework application, the `UnityModuleActivity`, which hosts the `UnityPlayer` instance referenced in the script above, implements the two methods seen below.

```
public void finishUnity(String resultMessage, String resultData)
public void updateUnity(String data)
```

Listing 3: Signatures of the two methods in the UnityContainerActivity that are called from the Unity script.

These two methods can then be called from the Unity script using the following two lines of code.

```
oActivity.Call("updateUnity", "data");
oActivity.Call("finishUnity", "message", "data");
```

Listing 4: Continuing from Listing 2, the oActivity object is used to call the Android methods shown in Listing 3.

The reason why the Unity program calls `Activity` methods rather than `Fragment` methods, as might be expected, is twofold. On one hand, it keeps the UnityScript code compact and clean as the developer does not need to know anything about `FragmentManagers` and `ViewPager`. On the other hand, `UnityModuleFragments` are integrated into the survey framework application in a different way compared to Android based modules. The reasons and technical details for this decision are outlined in detail in the next paragraphs.

The biggest challenge with adding support for Unity programs to the survey framework application was the synchronisation of a) the `UnityPlayer` instance running the Unity code, b) the `Activity` containing the `UnityPlayer` instance, and c) the process the `Activity` runs in. While synchronising the former two components is relatively straightforward, including the latter required some creative thinking. In older version of Unity the `UnityPlayer` class used to be its own independent class which, amongst other things, kept a reference to a View object that the `UnityPlayer` would use to draw its GUI in. In newer version of Unity the `UnityPlayer` is a subclass of Android's `View` class, allowing it to be directly attached to a layout. In addition to all the methods it inherits from the View class, it also has a few methods that help it synchronise with the Activity's life-cycle. Table 7 below lists the methods and the Activity's methods they need to be called from.

Table 7: Mapping between Activity methods and UnityPlayer methods. Calls to Activity methods are done exclusively through the Android system.

`UnityPlayer` method	`Activity` method	Description
resume()	onResume()	Informs the `UnityPlayer` that it can update itself to show the Unity GUI.
pause()	onPause()	Informs the `UnityPlayer` to pause execution as the Activity is currently not in the foreground.
onWindowFocusChanged()	onWindowFocusChanged()	Called when the current `Window` loses focus. Inherited from View.
windowFocusChanged()	onWindowFocusChanged()	Same as above. `UnityPlayer's` own method. Both methods must be called for correct functionality.
quit()	onStop()/onDestroy()	Informs the `UnityPlayer` to finish execution of the Unity program.

In general, the mapping from `UnityPlayer` to `Activity` life-cycle methods, as shown in Table 7, is simple, however it is easy to miss that the `UnityPlayer` class has two almost identically named methods, `onWindowsFocusChanged()` and `windowFocusChanged()`, both of which must be called for the `UnityPlayer` to function properly. The second peculiarity with respect to the `UnityPlayer` class is its `quit()` method. While it does not matter right away if the containing `Activity` is finished (by calling `finish()`) without calling quit() on the `UnityPlayer` instance, it appears to cause the `UnityPlayer` not to shutdown properly, which results in an application crash the next time a `UnityPlayer` instance is started. However, rather than just ensuring a proper shutdown of the `UnityPlayer` instance, `quit()` also kills its host process, resulting in the survey framework application to crash. So on first sight it seems that the only choice is between crashing now and crashing later. Fortunately however, Android allows applications to run more than one process. By default any component declared in an application's manifest runs in the same, default, process. A separate process for the component to run in can be specified by using the `android:process` attribute within a components declaration. In order to avoid killing the `Activity` which hosts all `Fragments` of a question group a separate `UnityContainerActivity` was declared and its `android:process` attribute set to `:unity_container`. The `Activtiy`/`Fragment` structure for Unity programs is shown in Figure 10.

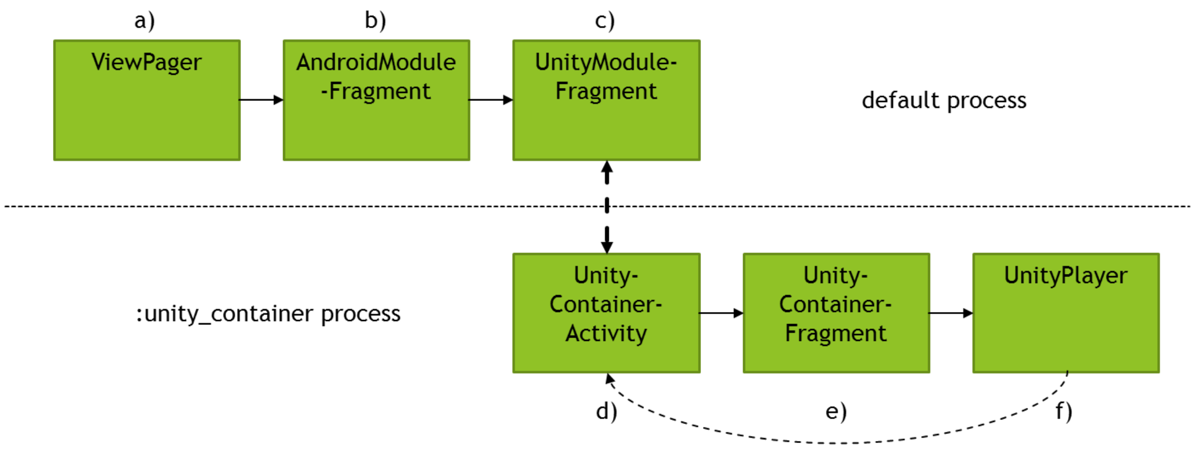


Figure 10: The Android application runs its components in two separate processes to prevent the UnityPlayer instance from killing the application when it completes.

The upper half of the figure is part of the structure previously shown in Figure 10 above, which runs in the default application process and contains the `QuestionGroupActivity` that holds the `ViewPager` (a) and the `Fragments` associated with the `QuestionGroup's` questions. Part (b) is the placeholder Fragment which loads a subclass of `UnityModuleFragment`. With a button press the application launches the

`UnityContainerActivity` (d), which is started in its own process as requested in the manifest declaration. To avoid recreation of the `UnityPlayer` instance (f) it is not directly inflated into the `Activity` but rather in the `UnityContainerFragment` (e) so that it can be retained across configuration changes. The aforementioned life-cycle mapping between the `UnityPlayer` and the `UnityContainerActivity` also had to be extended to the `UnityContainerFragment`. A fairly uncomplicated affair, as the `Fragment’s` life-cycle is tied to the `Activity’s` anyway. The execution of the Unity program within the `UnityPlayer` instance therefore happens in a separate process and `quit()` can be called safely without causing the application to crash.

While android offers a mechanism for inter-process communication (IPC) utilising remote procedure calls (RPCs), the execution logic for the Unity program is contained within the `UnityContainerActivity` and its subcomponents. Therefore the launching of the Activity and retrieving the experiment results from it can be easily accomplished using Android intent mechanism instead of IPC. `Intent` instances on Android can refer to their target component either explicitly by name or implicitly by the target’s capabilities. For components within the same application explicit `Intents` are the most common approach. For `Activities` in particular the methods `startActivity()` and `startActivityForResult()`, both taking an `Intent` object as their parameter, are used. As their names suggest, the former simply starts an `Activity` while the latter starts it, expecting a result from it. The latter was used to call the `UnityContainerActivity` (d) from the `UnityModuleFragment` (c). Before finishing the `Activity` all that needed to be done was to call its `setResult()` method, providing the result code indicating that the results are valid and the result data itself. This results in `onActivityResult()` to be called by the system for the original starting component, allowing the results to be passed on, to later be stored with the answers of the other questions.

This section showed how Unity applications can be loaded dynamically within the Android application and the interface both, the Android and Unity code, must conform to.

5.6 Deployment strategy

One of the goals envisioned for the system was to eventually make it accessible to other students across university departments to conduct their own surveys. The web interface for survey generation allows for basic survey to be generated including surveys with experiments. Despite the simplicity of the form based survey generation tool, technical knowledge is still required to implement an external question module. While no special tools are needed for the creation of a web module, a complete IDE is required for Android and Unity modules. In addition, access to some of the survey framework classes is required as well. The application code was therefore split into a regular Android project and a library project. The latter contains all classes needed to create the code for Android and Unity modules. Splitting the code into two separate entities has several advantages. Firstly, it reduces the size of the code, researchers need to include into their project, which ultimately affects the download time for the module. A reduction also means that researchers do not have to sift through over 4000 lines of code (LOC) to find the relevant classes and methods of the framework they need. Instead, they are presented with less than 800 LOC, which undoubtedly decreases the learning curve for any researcher, independently of their programming skills. Secondly, the split keeps parts of the framework private, allowing for changes to be made without having to worry that somebody might rely on a particular implementation detail that could change in future versions. The following deployment diagram shows which parts of the code base are packaged into the library project and which remain within the actual application. Furthermore it shows the components on the device as well as on the remote server.

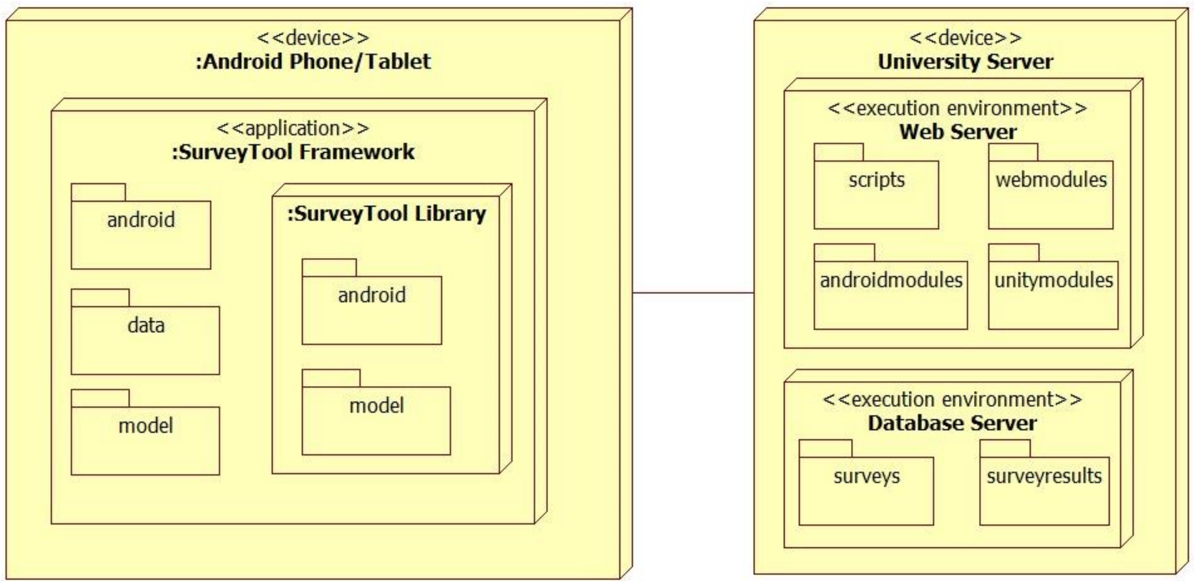


Figure 11: Deployment diagram for the survey framework. The only permanent component on the mobile device is the survey application itself. Third party modules are downloaded and executed as required and only stored temporarily on the device. The webserver provides the Android application with the necessary third party byte code and the researcher with an interface to create surveys. The database server stores the generated survey and any survey results uploaded from the Android device.

As can be seen on the left side of Figure 11, the survey application is deployed with the library binaries, which are also used when developing third party modules. The package names in the library and the application are abbreviated and have the common prefix *edu.ie.nuim.researchsurveytool*. They also contain sub packages that are not shown in the diagram. In general Android does not require the application to use the same package names as the library they use, however it was done in this case as the components belong to the same logical unit. On the right side of Figure 11 are the online components of the system. While the university server might actually comprise of multiple different actual and/or virtual machines, they are all administrated by the university and are therefore shown as a single entity. The web server component contains various directories which divide the system files into their relevant categories. The elements shown as packages within the database server component are the tables used to store system relevant information on surveys and their results.

5.7 Summary

This chapter gave a detailed insight into the concepts and classes underpinning the Android application of the survey framework. The model was discussed and it was shown how the two subclasses of the `QuestionGroup` class form the core of the model logic. Next the basic Android components `Activity` and `Fragment` were introduced and compared. It was shown that the `Fragment` class is better suited to hold the user interface for questions within a question group. Furthermore the more advanced component Android `ViewPager` was introduced. The advantages of using this component were explained as well as the additional implementation effort that was caused through a combination of the `ViewPager` component and the way the Android operating system handles the life-cycle of the two basic components `Activity` and `Fragment`. This concluded the introduction of all relevant components within the Android application.

The second part of the chapter analysed how code can be loaded dynamically in an Android environment. It showed what types of modules are supported and how web based modules and unity based modules have a similar interface to the Android application. Furthermore it introduced Android based modules and explained that Unity based modules use the same code loading technique. However, Unity based modules require additional code to access the assets and scripts of the Unity experiment itself.

Finally, the chapter shows how the components are deployed and what is necessary to allow external code to be actually downloaded and run on the Android application.

6 Example survey and System validation

This chapter describes the psychological experiment that was conducted to attempt to validate the viability of the survey framework. The question the survey set out to answer was PQ1: Is there a correlation in performance between the Stroop test and a vehicle following experiment? The purpose of the survey is twofold. Firstly, it aims to answer a research question in the area of psychology. Secondly, it shows how a survey might be conducted using the proposed framework to author and distribute the survey as well as collect the results and evaluate them. It is important to note that the main purpose of the conducted survey was to show the viability of the proposed survey framework. Therefore, the focus was on creating a survey that shows as many of the currently implemented features as possible and not to survey a statistically significant number of people. Since only 10 participants were surveyed the presented results are not statistically significant. However, the analysis was performed independent of that fact. Thus any claims in this chapter are not proven to be applicable to the general population, but do demonstrate the utility of the system framework to researchers.

Attention, in particular sustained attention, has been found to be a key factor in driving related tasks. A driver's attention on the road and traffic can be easily reduced, whether it is through the use of objects or engaging in non-driving related activities while manoeuvring a vehicle through traffic, lack of sleep, alcohol or drug consumption [35][36][37]. However, also demographic factors influence one's attention, such as age [38].

Sustained attention a cognitive function and can be measured with a variety of psychological tests. The Sustained Attention to Response (SART) test, the Simon task and the Stroop test are only a few examples [39][40][41]. The Stroop test was chosen for our experiment because it is a good measure of attention and has been used in several similar scenarios to assess participants' capabilities [42][43]. It is also generally used in driving related experiments and even in other smartphone-based tests [44][45]. A more general evaluation of the Stroop test is conducted by MacLeod [46].

The driving task builds on the work done as part of an undergraduate project at NUI Maynooth, in which a vehicle following experiment is designed [47]. The difference between the driving tasks lies within the user interface. The original experiment was designed for a desktop environment, with a full keyboard for user input. The experiment described in this report is conducted on a mobile the device, the user interaction with the virtual environment is facilitated through the devices touch screen. In order to avoid cluttering the relatively small screen of mobile devices with too many UI elements, the experiment was simplified by eliminating the need for the participant to keep the vehicle on a straight line.

The survey itself consisted of a set of 10 questions, asking the participants for demographic information and their driving experience. Table 8 lists the questions and their response type.

Table 8: Participants were asked to answer the 10 questions before they performed the Stroop test and the driving task.

Question	Type
Please indicate your gender:	Categorical
Please state your age:	Numerical
Please indicate your type of driver's licence:	Categorical
When did you obtain your current driver's licence?	Numerical
How many kms do you think you drive per year?	Categorical
Where do you typically do most of your driving?	Categorical
Do you have any medical conditions that could affect your driving? (Visual impairment, epilepsy, a heart condition, etc.)	Yes/No
Have you ever experienced motion sickness while driving?	Yes/No
Please indicate how you would rate your own driving ability, from 1 to 10, 1 being poor and 10 being excellent:	Numerical

6.1 Participants

This section describes the demographics and other important characteristics of the participants in this survey.

In our survey 10 participants were asked to answer the questions in the survey and to perform the Stroop test and a driving task. Out of all participants 7 were male and 3 female. All participants have an academic background, with 9 being students and 1 member of staff. The mean age of the participants was 22.7 (±2.5) years. Due to the relatively young age of the participants the variability of how long each of the participants has held their license for is quite small. One participant did not have a driver's license at all. The values range from 1 to 8 years, with an average of 3.8 (±2.4) years.

6.2 Stroop task

The participants were also asked to complete two tasks as part of the survey. Firstly, they were asked to complete a 90 second Stroop test, which measures the reaction time of participant to various types of stimuli. The stimuli are words displayed in different colours. The participant has to click one out of four buttons that corresponds to the colour of the word. The Stroop test is an established and widely used test in the fields of psychology and neuroscience [48][49][50][51]. The test in this particular survey contained congruent, incongruent and neutral stimuli which were occurring with a frequency of 50%, 25% and 25%, respectively. Congruent stimuli are words of colours written in the matching colour, e.g. the word "BLUE" written in blue colour. Incongruent stimuli are words of colours written in a different colour, e.g. the word "RED" written in green colour. Neutral stimuli are words that are not colours such as "BOOK" or "CAR. The distribution was chosen based on a Stroop task available in the PEBL test suit[5], a freely distributed collection of psychological test implemented in PEBL, the Psychology Experiment Building Language [52]. The participants had four buttons available to submit their response, where the button itself has a neutral colour and the button label is the name of the

[5] http://pebl.sourceforge.net/battery.html

associated colour. The colours of the labels themselves match the label. This is shown in Figure 12 which shows a screenshot of the Stroop task on a mobile device.

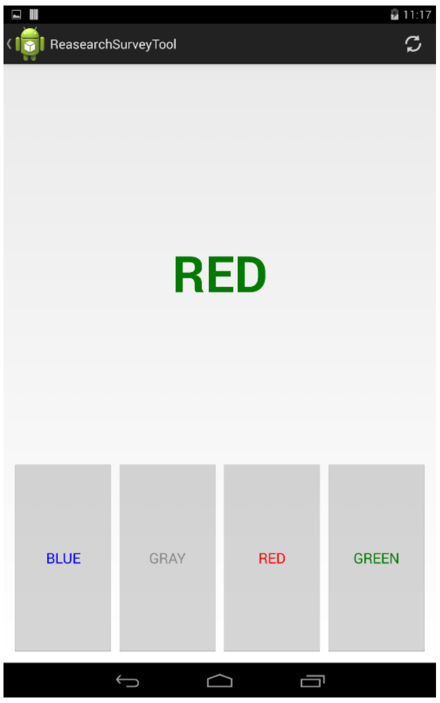


Figure 12: The Stroop task as seen by the participant on the mobile device. An incongruent stimulus is shown and the participant has four possible responses. In this case the correct answer is "GREEN".

The event based data from the Stroop task was recorded as tuples of timestamp and event. The five possible events for the Stroop tasks are split into two groups, "show" events and "response" events. The "show" events are split into "show congruent", "show incongruent" and "show neutral", whereas the "response" events can only take the value "correct" or "incorrect". Together with the timestamp the data can be ordered chronologically and analysed. Table 9 shows the measures that were calculated from the Stroop task data for each participant.

Table 9: The 13 measures that were calculated for each participant's Stroop task results. The sum of the first 6 measures shows the total number of interactions the participant had during the task. Of particular interest are the incorrect responses to incongruent stimuli as they indicate a lapse in attention.

Measure	Description
cong-c	Number of correct responses for congruent stimuli
cong-inc	Number of incorrect responses for congruent stimuli
incong-c	Number of correct responses for incongruent stimuli
incong-inc	Number of incorrect responses for incongruent stimuli
neutral-c	Number of correct responses for neutral stimuli
neutral-inc	Number of incorrect responses for neutral stimuli
stroop-score	Total number of correct responses
rt-avg	Average reaction time per stimulus
rt-avg-c	Average reaction time for correct stimuli
rt-avg-c-cong	Average reaction time for correct congruent stimuli
rt-avg-c-incong	Average reaction time for correct incongruent stimuli
rt-avg-c-neutral	Average reaction time for correct neutral stimuli
multiple-tries	Number of stimuli the participant responded to incorrectly more than 2 times

6.3 Driving task

The second experiment the participants were subjected to was a virtual driving task. The participants were asked to follow the car on the screen in front of them, while maintaining a constant safe distance. Only two buttons, “ACCELERATE” and “BRAKE” were available to the participants. The road geometry is a straight line and there are no trees, houses or traffic signs on the side of the road. The user interface is shown in Figure 13. The experiment duration was set to 190 seconds, with only the last two minutes being evaluated in the analysis. The first 70 seconds were used to give the participants a chance to familiarise themselves with the user interface and were therefore discarded. During the experiment the lead car would brake 7 times at random times (real brakes). Furthermore the brake lights of the lead car would switch on 3 times without the car actually braking (fake brakes). This results in a total of 10 brake event perceived by the participants. By analysing the difference in reaction to real and fake brakes a conclusion can be made whether participant react to the brake lights or rather the distance to the lead car.

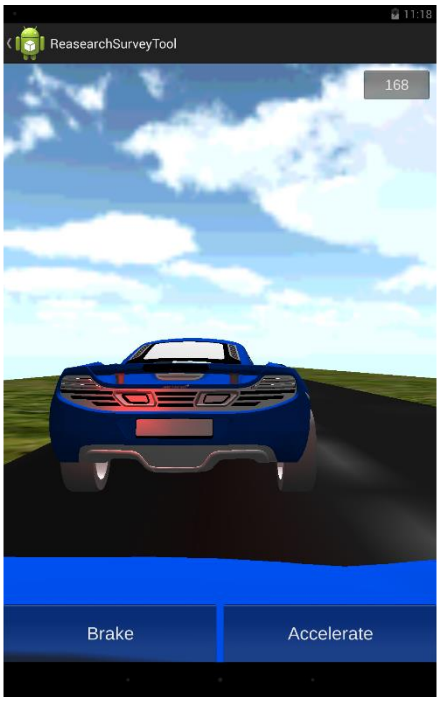


Figure 13: The user interface for the driving task. The participant can apply the brake using the left button or accelerate using the right button. If neither of the buttons are pressed the car will start to decelerate. The timer in the upper right corner shows the participant when the experiment will finish.

The data was recorded by sampling 4 measurements every 0.3 seconds. The measurements recorded the time, the state of the user's car, and the state of the lead car and the distance of the user to the lead car. The user's car could be in one of three states, namely "ACCELERATING", "DECELRATING" and "BRAKING". The lead car however, has an additional, fourth state, which is "BRAKE LIGHTS", indicating that the car's brake lights are switched on without the car's brakes actually being applied. Similarly to the Stroop task, by ordering the samples chronologically the measures shown in Table 10 can be calculated.

Table 10: The 8 measures calculated from the driving task data for each participant.

Measure	Description
dist-avg	The average distance between the two cars
dist-sd	The standard deviation of the distance between the two cars.
user-acc	The percentage of time the user spent accelerating
user-dec	The percentage of time the user spent decelerating
user-br	The percentage of time the user spent braking
rt-br-avg	The average reaction time to a lead car brake
missed-br	Number of times the user did not react to a lead car brake
missed-br-safe	Number of times the user did not react to a lead car brake but maintained a safe distance.

In addition to these calculated measure, the raw data can also be visualised to get a quick understanding of how a participant performed. A diagram showing a test run of the experiment is shown in Figure 14.

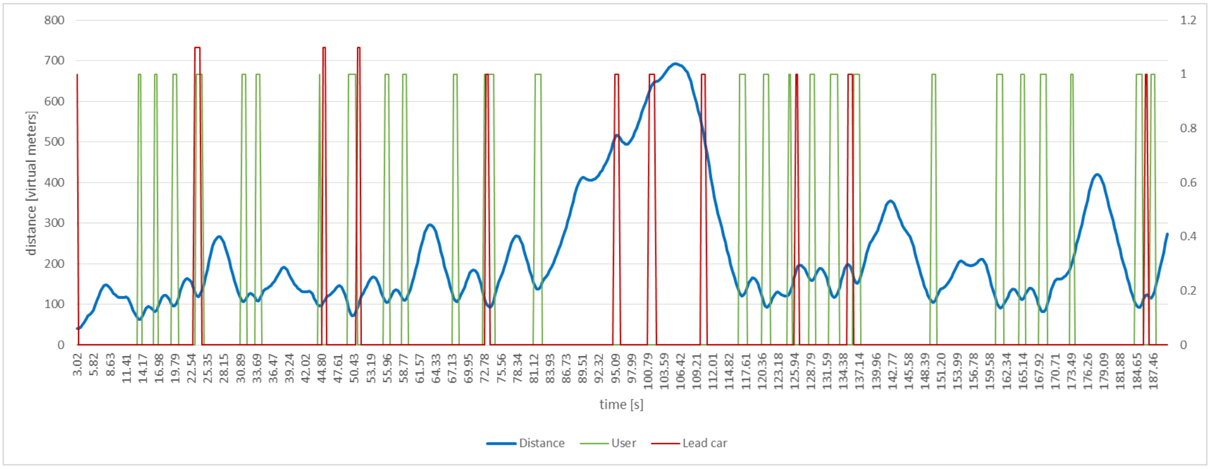


Figure 14: Sample data from a driving task. The blue line shows the distance between the two cars over the course of the experiment. Its values are shown on the primary vertical axis (left). The green line shows when the survey participant was braking. The red line indicates brakes of the lead car. For the lead car fake brakes are also shown. In this example they occur in the first third of the experiment and have a value of 1.1 compared to real brakes which have the value 1.

It is important to note that the graphical assets, i.e. the road and the cars were used as they were provided. One distance unit within the provided model represents 0.2m.

6.4 Analysis

This section describes the finding from the analysis of the gathered results. Data from 10 participants was used in the analysis. Every step is done programmatically to ensure full reproducibility of the presented results. First, some general observations about the participant data are discussed. Secondly, the two experiments are compared and the correlations between the measures of the Stroop and the driving task are analysed.

6.4.1 General observations

Research on road accidents shows that young and male drivers are more likely to be involved in accidents [53][54][55][56]. Since gender and age were recorded for the participant of this survey, the data was examined to see if gender has an impact on key measures of the driving task. Figure 15 show box plots for the average reaction time to a lead car's brake, the average distance between the participant and the lead car and the standard deviation of the distance, respectively, for male and female participants.

It is important to note that this section shows how possible conclusion could be drawn from the presented survey. It is intended as a pilot study and has no statistically significant results. However, a future collaboration with the Psychology department here at NUI Maynooth is intending to repeat a revised version of the described survey with a larger number of participants to create statistically significant results.

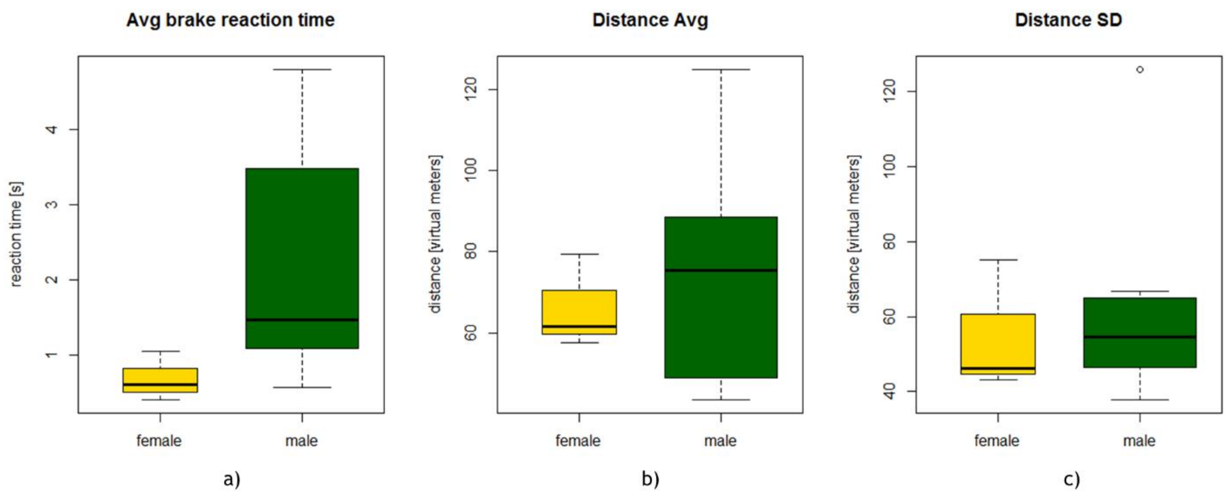


Figure 15: The three key measures for the driving task as box plots for females (yellow) and males (green). Each box represents five pieces of information. The whiskers represent the lowest and highest values with each group that are not outliers. The box itself represents the range of values from the first to the third quartile within a group. The bold black line within the box indicates the median value within the group. Outliers are shown as small circles. Values are considered outliers if their distance from the box is greater than the box height.

The results in Figure 15 a) seem to indicate that females in the test group tended to react much quicker to a brake of the lead car. Furthermore, there is a much higher variability of reaction times with the male participants than there is with the females. The data gathered in this experiment perhaps indicates that female drivers are generally more attentive, or that female participants are simply more cautious when they see a car brake in front of them. Since the participants were not asked to brake as soon as the lead car brakes but rather to maintain a constant safe distance, the high reaction time with the male participants could indicate that they are less exact when keeping a constant distance. This theory is supported by the data on the standard deviation of the distance as shown Figure 15 c), which shows that only a minority (<25%) of male participants have a smaller variability than a majority of the female participants, i.e. only few male participants are better than most female participants at keeping a constant distance.

Another interesting observation is that the male participants seem to disagree on what constitutes a safe distance. As indicated by the data shown in Figure 15 b), the variability of the average distance within the group of male participant is much higher compared to the female group.

However, when after further examining the data it becomes apparent that none of the results were statistically significant, as measured with the t-test. The results are shown in Table 11 In addition, not only is the sample size rather small with only 10 participant, only 3 participants were female. This makes it very difficult to generalise to the general population. As the results are not significant it cannot be said using this experimental data that females were more cautious or attentive than their male counterparts.

Table 11: Results of significance tests for gender related differences. Significance tests are a statistical measure expressing whether the results of an experiment have occurred by chance or whether they show a pattern. The t-test used determines whether the means of two sample groups are different enough to allow a conclusion to be derived based on that difference. The t-values indicate the similarities between the groups. The bigger the - value, the bigger the difference between the groups. Additionally, the p-value indicates how likely it would be to get their corresponding t-values from random data. A p-value of 0.882 means that there is an 88.2% chance that the difference between the two groups occurred by chance. Therefore, the smaller the p-value the more significant the results. In general, results are said to be significant when the p-value is below 0.05, thus the results shown in Figure 15 are not statistically significant. The experiment would therefore need to be continued until a significant result is achieved, i.e. the p value drops below 0.05.

measure	t	p
Average brake reaction time	0.153	0.882
Average distance	0.676	0.516
Standard deviation from average distance	0.671	0.519

6.4.2 Experiment correlation

Since the data set collected is not very big and the number of measures is relatively small, the correlations between all numeric measures were calculated to identify pairs of measures that might be of interest. The full table of correlations can be found in Appendix A. This section only covers the most interesting correlations, relevant to the above research question PQ1.

Firstly, the data shows a strong correlation between the individual measures of the Stroop test. This simply shows that the different measures are all related, e.g. the number or correct congruent trials is related to the total number of congruent trials. The same inter correlation is found with the measures of the driving task.

Secondly, due to the small number of participants the dataset does not contain a single missed brake and only one safe missed brake. When a participant did not react to the brake lights of the lead car it was considered a missed brake. However, if the distance between the participant and the lead car was greater than their average distance plus one standard deviation, the missed brake was regarded as safe. Because only safe missed brakes were recorded for the participants, it can be said that none of the participants had significant lapses in attention while performing the driving task. Therefore any correlation with the measures “missed brake” was ignored.

The correlations between measures of the Stroop task and the driving task have an overall strong correlation. In particular the average distance and the standard deviation from the average distance have very strong correlations with almost all the Stroop task measures. Especially, the correlation to the reaction time measure of the Stroop task has absolute values of 0.73 to 0.91. This indicates that the performance in both tasks are somewhat related.

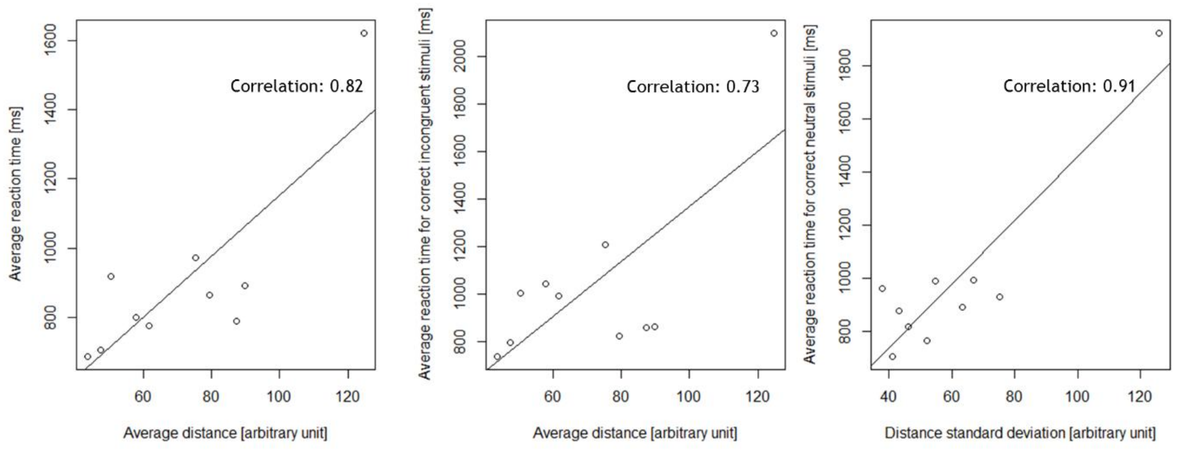


Figure 16: Reaction time measure of the Stroop task and the driving task show strong correlations, with their absolute values ranging from 0.73 to .91.

Other correlations with lower values include the questions with numerical responses such as the number of years a participant has had their license or their age. This is probably in part because of the limited age range of the sample population. An interesting correlation however was found between the duration for which a participant has had their license and their perception of their own driving skills.

6.5 Summary

The survey framework was used to create and conduct a survey that attempts to correlate the participants' performance in two tasks. The first is the widely used Stroop task which measure reaction time and sustained attention. The results from the Stroop task are compared to the results from a vehicle following experiment, where the participant is asked to maintain a constant safe distance to a car in front of them. The survey creating and delivery worked as expected and delivered the raw data to the scientist. The subsequent evaluation indicated that there are strong correlations between reaction times in the driving task and the reaction times in the Stroop task. Unfortunately, due to the small number of participants, the results are not statistically significant.
In summary it can be said, that this area of research needs further investigation, since there seems to be a correlation between the two experiments. However, it is imperative to re-run the experiments on a larger scale with a much larger number of participants.

From a software engineering perspective, the experiment demonstrated the feasibility of research surveys with the proposed framework. Especially due to the mobility and flexibility of mobile devices, the survey could be easily repeated with more participants.

6.6 Threats to validity

The main issue with this survey is the small number of participants and the fact that the sample was drawn almost exclusively from the international student population. This means that similarities between participants are much stronger than in would normally be in the general population. In particular this results in a narrow age range and a narrow range for how long participants have had their current license. Also, while most participants had a valid license they did not drive a car in the last few months, which could potentially affect the results of the driving task.

From the perspective of measures and the evaluation of the raw data, a more sophisticated approach to what constitutes a missed brake could have been employed. However, since the number of missed brakes and safe missed brakes is very small this does not affect the evaluation of the particular dataset analysed in this report.

7 System verification

In this chapter we analyse and evaluate the system. Unlike conventional commercial software products, the verification focus for the proposed survey framework is on system testing rather than unit testing. While unit testing is an important aspect of software development, in a research project like this it is more important that verification is performed on the experiment results. However, one vital data model component of the system was unit tested. This section describes system testing that was conducted to verify that the results gathered during the survey presented in chapter 6 are correct. The chain of dependencies is analysed to determine if the aspects of the systems that must function properly in order to ensure the correctness of the data gathered by the survey framework. Moreover, the unit testing techniques used to verify the functionality of a key component in the system are shown. Finally the limitations of the current implementation are discussed.

7.1 System testing

While, in an ideal world, the entire system would be unit tested, testing has to be prioritised in a time constraint project such as this. Therefore, testing focuses on verifying that the system produces the correct output which is then analysed by researchers and used to accept or reject a given hypothesis. This requires a chain of dependencies to be identified and verified. Each link in the chain takes some input, processes it and passes its output on as input to the next link in the chain. This is depicted in Figure 17. This section shows how the individual links were verified, or if they were not verified, how they would be.

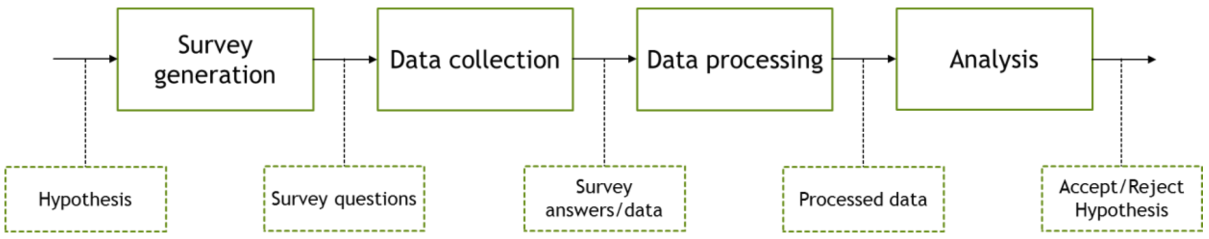


Figure 17: Chain of dependencies. Each link has receives input that it processes. The resulting output is relayed as input to the next link. Each link must be verified to ensure that the output matches the expected input into the next link. E.g. if not all questions the researcher adds to a survey are displayed on the participants screen, any further action will inevitably lead to the wrong data being delivered back to the researcher.

7.1.1 Survey generation

Surveys are generated using a web based user interface which was built using common web technologies such as HTML, CSS and Javascript. Once the survey is submitted it is stored in a database using PHP and PostgreSQL. Again, using PHP the survey can then be retrieved and displayed on a mobile device. Manual testing was conducted to ensure the proper functioning of those components. Initially this was done explicitly during the development of the components and later implicitly by conducting the survey presented in chapter 6. In order to properly verify each component unit testing could be used to test that data that is being submitted is entered into the database. Moreover, it could be tested that the survey retrieval mechanism returns the correct survey that was requested based on its ID. Furthermore, the web UI could be tested using a software testing framework for web applications such as Selenium[6]. Selenium automates browser testing by directly interacting with the web application as displayed in the browser. While this technique is not as fast as unit testing, it does enable test to be written based on use cases, such as creating a survey with one question of each question type.

[6] http://docs.seleniumhq.org/

7.1.2 Data collection

Data collection is the process in which survey participants respond to survey questions or perform experiments. The results are textual, numerical or categorical answers to the questions and raw or pre-processed experiment data. The main components involved at this stage are the Android application and any third party modules that are loaded with the survey. As such, the verification can be split into sub activities. Firstly, the framework should be verified by checking that all the questions within a survey are presented to the participant. Furthermore, it must be ensured that the survey module download and survey result upload function properly. That is, all the data must be transmitted to and from the server and must not be altered during transmission. While this step was only verified manually by running several demo surveys and uploading their results, it could be formally checked using some form of hashing. For example, a checksum could be generated for the survey and its modules, which is compared to the checksum of the downloaded files on the device. Analogously, a checksum could be created and sent to the server which only accepts the results if their checksum matches the value that was sent by the remote device.

The second aspect that must be verified is the correctness of any externally loaded modules. A programming fault could immediately render all collect data useless, depending on the type of fault. Ideally, external modules would be unit tested. However, it requires substantial time and effort to properly unit test UI components, especially when there is a timing component involved. Therefore, rather than attempting to verify a small portion of the module rigorously, more general sanity checks were carried out. For the two external modules presented in the example survey in chapter 6, it was checked whether the reaction times reported for both tasks are correct. Instead of trying to accurately compare the measure reaction with the actual reaction time, an experiment was performed to determine whether the values are within the expected range. For both modules, very low, low and high values were checked. The concrete values and the average results of the performed trial runs for each module are shown in Table 12.

Table 12: The average reaction time measures for the two external modules are compared to their expected values. The average result is based on 9 - 17 sample measurements for each category.

Expected time	Stroop task	Driving task
~150ms	149.412ms	1.24s
~1s	1.012s	1.206s
~10s	10.006s	10.021s

The experiment was conducted by manually clicking on buttons every, 10 seconds, every second, or as fast as possible. Then the values recorded by the system are compared to the expected results. For the Stroop task the actual values very closely match with the expected values. Conducting the same experiment for the driving task however shows that, while the correleation is good for 10 second and 1 second intervals, the recorded time for very fast interactions deviates significantly. This is due to the fact that the data from the experiment is recorded 3 times per second, resulting in a resolution of 0.333333 seconds. The implication is that the granularity of reaction time measurements is 0.333333, which should be improved when the experiment is run again. Increasing the sampling frequency would simply mean that more data needs to be transferred to the web server. This is well within the capabilities of the system. The maximum sampling frequency that can be achieved depends on the maximum frame rate of the device for the experiment. Theoretically, measurements could be made up to 60 times per second. Alternatively, the data processing could be done during run-time instead of sending the raw data to the researcher to do the processing offline.

The other measure that was checked for the driving task, is the distance between the user and the lead car. While the distance is not reported in meters, but rather in a virtual distance (as explained in section 6.3), the value should increase proportionally to the lead car's speed if the user car remains at the starting position. Therefore, the lead car was set to drive without braking and at a constant speed for 20 seconds, while the user car does not move at all. The data is then checked if the reported distance increases linearly. Figure 18 shows the result of this test.

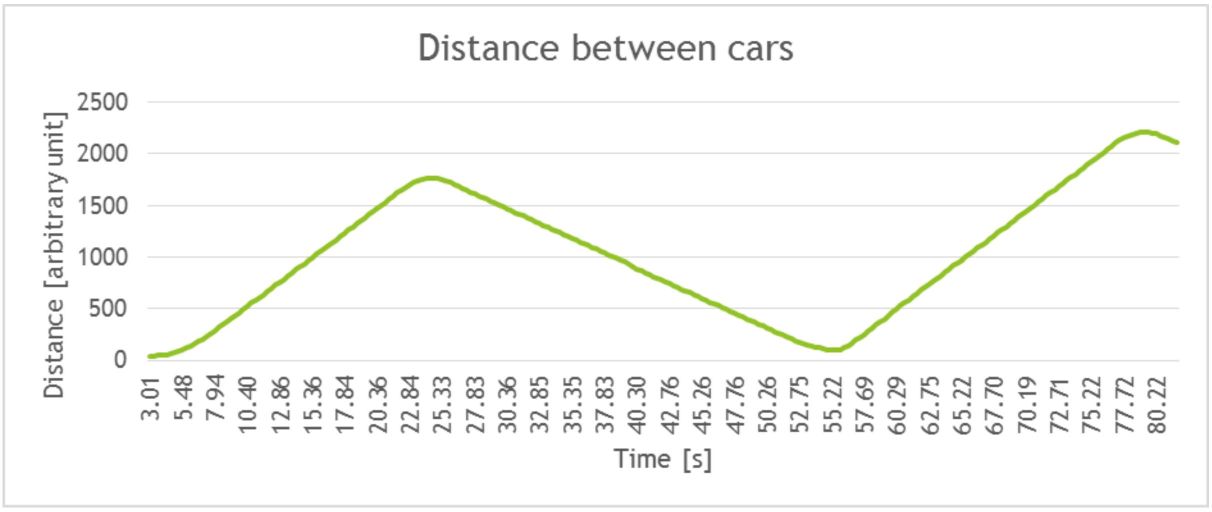


Figure 18: Testing the distance between the two cars. Initially, the user car remains in its starting position and the lead car accelerates up to a constant speed. Then the user car accelerates to maximum speed. This is repeated twice.

After the lead car reaches its constant speed around 6 seconds into the experiment the distance increases linearly up to the point where the user car is accelerated to maximum speed. The speed of the user car is greater than the speed of the lead car and therefore the distance between the two cars decreases. This shows that the reported distance is indeed correct.

7.1.3 Data processing & Analysis

In general, data processing can be implemented within the external module. Alternatively, as in the case of the example survey, raw data is sent to the server, in which case the data processing occurs separately. Ideally it would be subjected to the same type of unit testing covered in the previous sections. Because of the research oriented nature of the project functionality was prioritised over detailed unit testing.

The analysis part is a manual task, as the processed data needs to be interpreted by a human expert. In the case of the example survey, the entire analysis process is described in the previous chapter.

7.2 Unit testing

Ensuring that the low level components of an application behave as expected is the key requirement to the overall correct functionality of a system. There exist several black box and white box techniques that help developers to properly test their code by comparing expected output to the actual output of their code for a carefully selected set of test cases. Those techniques are in place because exhaustive testing, i.e. checking the output of a method for all possible inputs is often not feasible. Rather than spending valuable time on unit testing the complete system, system tests ensure that the data produced by the experiments is valid. One of the core components of the data model is unit tested. The following section shows the unit testing conducted for the class `RecurringQuestionGroup`. The same techniques could be applied to any other class within the framework.

7.2.1 RecurringQuestionGroup testing

This section shows the testing conducted to ensure the correctness of the `RecurringQuestionGroup` class. It is a concrete implementation of the abstract class `QuestionGroup` and can contain subgroups of type `RecurringQuestionGroup`, as well as a list of `Question` objects. In addition it holds a `Timing` object which specifies during which times a participant can be asked to answer the `Question`s within the group and its subgroups. A `RecurringQuestionGroup` is said to be independent if and only if its `Timing` object is not `null`.

Testing of the class is split into black box testing, where the expected output is only based on the specification of the class, and white box testing which ensures that all parts of the class are exercised. All testing techniques are taken from “Software Testing - Principles and Practice”, by Brown et al. [57]. Testing this class is relevant because the `QuestionGroup` class is at the core of the survey framework data model. Since the abstract class cannot be tested directly, the choice is between `OneTimeQuestionGroup` and `RecurringQuestionGroup`. The latter was chosen because it subsumes the functionality of the `OneTimeQuestionGroup` which does not add any extra functionality to what it inherits from `QuestionGroup`.

7.2.1.1 Black box testing

This section shows how black box testing was conducted for the non-trivial methods in the `RecurringQuestionGroup` class. One line setter and getter methods are regarded as trivial, and are therefore not described in this report although there were implemented. Table 13 lists all non-trivial methods that were tested for the class under test. Since we are testing objection oriented software, the appropriate techniques must be used. This section shows how equivalent partitions (EP) and boundary value analysis (BVA) were conducted in a class context.

The most complex method within the class is `getQuestionAt()` which returns a `Question` object based on its position within the group or its subgroups. Since the Equivalence Partitions (EPs) and therefore also the Boundary Values (BVs) depend on the number of subgroups, their nesting and the number of questions contained within each subgroup, testing was performed by selecting a representative example. In particular 4 cases with different processing were identified.
Firstly, a question group can only contain questions and no subgroups. Secondly, a group can contain a single sub group with some questions. Thirdly, the group's subgroup itself can contain a subgroup. And finally, the question group can have multiple subgroups with some of them having subgroups themselves.

These four cases can be summarised in one example which is displayed in Figure 19. The number of test cases in order to achieve exhaustive testing is small as one test case is required for each question within this structure. The resulting 19 test cases were implemented and uncovered two faults which were subsequently corrected.

Table 13: List the methods that were tested for the class, including their parameters, return values and a brief description.

Method name	Parameter	Return value	Description
getQuestionCount	-	int	Returns the number of questions within the group (including sub groups).
getQuestionAt	int	Question	Returns a Question based on its position in the group (or subgroups).
getQuestionById	int	Question	Returns a Question based on its unique identifier.
addSubGroup	Recurring-Question-Group	-	Adds a subgroup to the group and sets the subgroup's owner to be this group.
addQuestion	Question	-	Adds a question to the group and sets the question's owner to be this group.
getDuration	-	int	Returns the duration in minutes it takes for a participant to answer all questions within this group (including subgroups).
getAllIndependent-SubGroups	-	List<Recurring-QuestionGroup>	Returns a list of sub groups of this group that are independent, i.e. have their own Timing object.

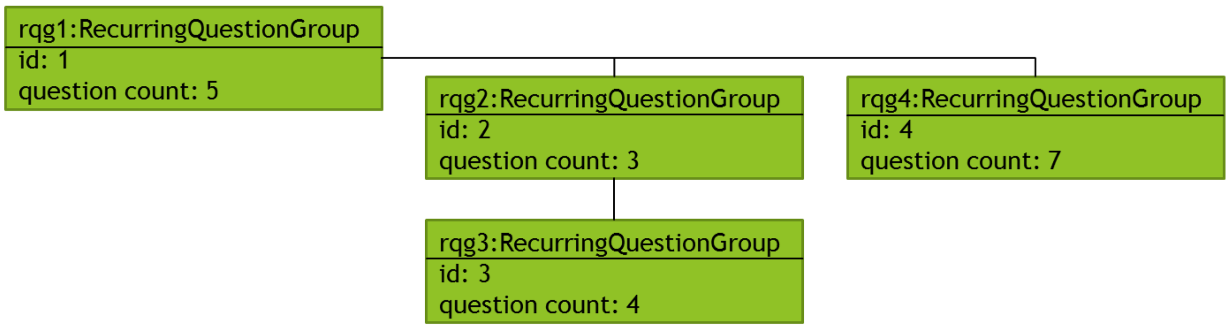

rqg1:RecurringQuestionGroup
id: 1
question count: 5
rqg2:RecurringQuestionGroup
id: 2
question count: 3
rqg4:RecurringQuestionGroup
id: 4
question count: 7
rqg3:RecurringQuestionGroup
id: 3
question count: 4

Figure 19: Sample question group structure.

For a more complicated structure the EPs BVs would need to be identified and tested. In the above example the EPs are related to the number of direct question within a group and the structure of the groups. The EPs are summarised in Table 14.

Table 14: Equivalence Partitions for the example question group shown in Figure 19. The first and last EPs are error cases. The BVs are the start and beginning of each EP.

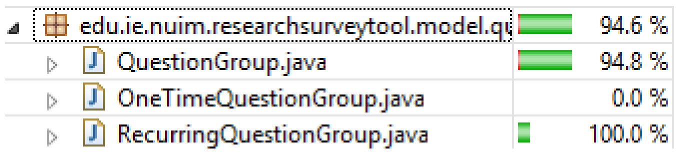
MIN_VALUE..-1	0..4	5..7	8..11	12..18	19..MAX_VALUE

The remainder of the methods was tested without formally specifying EPs and BV because their logic was fairly straightforward. However, he same technique as described for the method `getQuestionAt()` could have been applied.

7.2.1.2 White box testing

White box testing was conducted on the `RecurringQuestionGroup` class to ensure statement coverage. The coverage tool used was Ecl Emma[7]. As can be seen in Figure 20, the test for the `RecurringQuestionGroup` class also cover most of the `QuestionGroup` class. The only part that was not covered was an empty, private, no-args constructor which is needed for an external API that was used in the project.

[7] http://www.eclemma.org/



edu.ie.nuim.researchsurveytool.model.qu	94.6 %
QuestionGroup.java	94.8 %
OneTimeQuestionGroup.java	0.0 %
RecurringQuestionGroup.java	100.0 %

Figure 20: Snippet of the Ecl Emma coverage report for the class RecurringQuestionGroup.

No test were required in addition to the black box tests to achieve this coverage.

7.3 Limitations

In addition to the limitations mentioned in chapter 5, there are a few issues with the current implementation which would stand in the way of a public release. This section briefly gives an overview of those issues.

The biggest limitation is that the framework does not allow the user to utilise one of the most useful features the Android platform has to offer. Loading pre-compiled resources, which are packaged in the *resources.arsc* file of an Android application file is not possible with external code modules. This requires the entire layout of the experiment to be created with the appropriate API in Java. It also restricted binary resources such as image or audio files to be accessible by placing them on a server and downloading them the first time the experiment is run.

While the system model allows recurring question groups to be included in a survey, the current implementation of the Android application and the online survey generation tool, does not support this feature. The online tool would need to be adapted to support nested groups and the ability to reference groups in multiple locations to allow researchers to fully take advantage of the system model. The Android application already supports nested groups to a certain extent. The bigger implementation detail that is missing is the notification system that would remind participants to answer recurring questions.

What is more, the access to the server and thus the survey data is not restricted. Since the focus of the project lies elsewhere the server component would require a security overhaul before releasing the system publicly.

8 Future work

This chapter shows that the current implementation of the system is merely a stepping stone to a much more comprehensive surveying application. The current implementation of the system achieved its goal, which was to create a general purpose survey tool with support for external survey experiments. The potential future features discussed in this section show what is required to make the system of practical use for researchers, especially a researcher with little technical background. Furthermore potential functionality building on the existing code base is proposed.

The first thing that would need to be done before the public can be given access to the system is to put proper security measures in place, to protect the privacy of the data submitted by the participants but also to allow researchers to keep their research private until it is ready to be published. In order to achieve this, registration and login functionality would need to be added to the web server component. Also the data should be stored in an encrypted format. Furthermore, despite the fact that transmission between the mobile application and the server are performed using the HTTPS standard, the data itself should be also encrypted within the local device database and transmitted in this form to ensure that nobody intercepting the data in transit can misuse it.

Similarly, survey participants could be asked to register to ensure that questions that are supposed to be only answered once per participant are not answered multiple times. This would require some effort as it would need to ensure that the data submitted by the participant still remains anonymous.

A survey related feature available in many of the online survey tools researched is the capability of defining a skip-logic. This allows the survey tool to skip questions based on the participant’s answer of a previous question. This could eliminate questions such as “If you answered the previous answer with yes, please answer the following…”.

As mentioned in the section Question groups, the model supports recurring question groups. However, this functionality was not implemented for the Android application or the survey generation tool. Adding this functionality would greatly extend the number of types of survey researchers could conduct.

Another security related potential feature would be a code checker which performs static analysis on the third party modules. Currently, the survey tool requires only a limited number of permission such as internet access from the Android system. When a third party module requires access to the camera for example, the application would crash with a security exception as the survey tool does not have permission to access the camera. Static analysis could be performed to at least prevent modules from being executed if calls to API are found that the survey tool has no permission to access. Going a step further such analysis could be done when researchers attempt to generate a survey that includes a problematic module. This would give them feedback much earlier in the development process.

In addition to supporting Android, Web and Unity modules, the inclusion of other frameworks and game/physics engines could yield a great benefit to the usability of the system. In particular it would be of interest to add module integration for frameworks using other programming languages than Java, C# and Javascript. Adding support for e.g. python modules would allow the research community to reuse any experiments written in python and greatly reduce the porting effort that would be required to make it run within the survey tool. In fact, existing work in this area could be used to extend the supported module types [58].

A current limitation of the survey tool is that Android based modules cannot use precompiled resources such as xml layouts, nor binary assets such as images or audio files. Instead layouts must be defined programmatically and binary assets must be loaded at run-time from a remote location. This not only reduces the legibility of the code but is also a much more cumbersome way of specifying a layout in Android. Adding support for such resources would greatly increase the code quality of third party modules and would reduce the burden of development on the researchers.

Static analysis could also be performed to identify any online resources that third party modules might need, allowing the system to download them when the survey is first added to the application. This would allow the survey tool to work without any internet connection once the survey is added.

9 Conclusion

In a world driven by big data, it is imperative that researchers adapt their methods to the available technology that allows them to scale their research. This report discussed a framework that incorporates many aspects of today's technology to allow researchers to conduct their survey in a much more automated and scalable way.

The proposed system implements a generic survey framework with support for external code modules which can contain custom question types or experiments. While the application is developed for the Android platform, third party modules can be Android, Web or Unity modules. An online survey generation tool allows researchers to easily create new surveys. The system overview and implementation details showed the underpinnings of the system which was built using Java, JavaScript, HTML, CSS, PHP, UnityScript, SQLite, PostgreSQL, JSON and XML.

To prove the feasibility of the system a psychology survey was conducted investigating the correlation between the well-known Stroop task and a driving task where the participants are asked to maintain a constant safe distance to the lead vehicle. The results did show a correlation between the measures of both tasks. However, due to the limited number of survey participants further investigation is required to get a conclusive result. More importantly however, the process of how a survey would be conducted using the proposed system was shown, and thereby validating the system.

Finally, the system is verified using a number of system tests. System testing was conducted to ensure that the data reported by experiments is valid. Detailed unit testing was performed on a critical component of the system.

We believe that the proposed system can be taken even further and that it could greatly increase the impact of the work of researchers.

10 References

[1] SurveyMonkey, '"SurveyMonkey: Free online survey software & questionnaire tool:", vol. 2014, no. 20 May 2014.

[2] KwikSurveys, '"KwikSurveys: Free online survey & questionnaire tool ", vol. 2014, no. 20 May 2014.

[3] Google Inc, '"Home | Google Consumer Surveys:", vol. 2014, no. 20 May 2014.

[4] R.P. Pargas, J.C. Witte, L. Brand, C. Hochrine and M. Staton, '"OnQ: an authoring tool for dynamic online surveys," *Information Technology: Coding and Computing [Computers and Communications], 2003. Proceedings. ITCC 2003. International Conference on*, pp. 717-723.

[5] J. Burkey and W.L. Kuechler, '"Web-based surveys for corporate information gathering: a bias-reducing design framework," *Professional Communication, IEEE Transactions on*, vol. 46, no. 2, pp. 81-93.

[6] A. Singh, A. Taneja and G. Mangalaraj, '"Creating online surveys: some wisdom from the trenches tutorial," *Professional Communication, IEEE Transactions on*, vol. 52, no. 2, pp. 197-212.

[7] J. Kite and Leen-Kiat Soh, '"An intelligent survey framework using the life events calendar," *Electro Information Technology, 2005 IEEE International Conference on*, pp. 6 pp.-6.

[8] A. Attarwala, A. Das and D. Wilson, '"Mobile Platforms: A New Frontier for Market Research," *Mobile Services (MS), 2012 IEEE First International Conference on*, pp. 115-116.

[9] C.R.K. Stradiotto, A.I. Zotti, C.O. Bueno, S.P.M. Bedin, H.C. Hoeschl, T.C.D. Bueno, T.P.S. Oliveira and V.O. Mirapalheta, '"2010 IEEE International Conference on Progress in Informatics and Computing; Web 2.0 e-Voting system using android platform ", pp. 1138 <last_page> 1142.

[10] Dooblo, '"Android Survey App | Conducting Surveys using Android Survey App ", vol. 2014, no. 20 May 2014.

[11] CREOSO Corp, '"Rollapoll mobile survey app for Android tablets ", vol. 2014, no. 20 May 2014.

[12] SurveyPocket, '"SurveyPocket - Home,", vol. 2014, no. 20 May 2014.

[13] QuickTapSurvey, '"Mobile & Offline Survey App ", vol. 2014, no. 20 May 2014.

[14] Li Gong, '"Secure Java class loading," *Internet Computing, IEEE*, vol. 2, no. 6, pp. 56-61.

[15] L.L. Petrea and D. Grigora, '"Dynamic Class Provisioning on Mobile Devices," *Parallel and Distributed Computing, 2006. ISPDC '06. The Fifth International Symposium on*, pp. 140-147.

[16] Jin-Cherng Lin and Jan-Min Chen, "'The Automatic Defense Mechanism for Malicious Injection Attack," *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, pp. 709-714.

[17] Jin-Cherng Lin, Jan-Min Chen and Cheng-Hsiung Liu, "'An Automatic Mechanism for Sanitizing Malicious Injection," *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, pp. 1470-1475.

[18] M. Payer, T. Hartmann and T.R. Gross, "'Safe Loading - A Foundation for Secure Execution of Untrusted Programs," *Security and Privacy (SP), 2012 IEEE Symposium on*, pp. 18-32.

[19] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel and G. Vigna, "'Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications,".

[20] D. Sbirlea, M.G. Burke, S. Guarnieri, M. Pistoia and V. Sarkar, "'Automatic detection of inter-application permission leaks in Android applications," *IBM Journal of Research and Development*, vol. 57, no. 6, pp. 10:1-10:12.

[21] H. Shukla, V. Singh, Young-Ho Choi, Jaeook Kwon and Cheul-hee Hahm, "'Enhance OS security by restricting privileges of vulnerable application," *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on*, pp. 207-211.

[22] S. Hatwar and C. Shelke, "'An Assess Android Antimalware that Detects Malicious Dynamic Code in Apps," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 3, pp. 263.

[23] B.J. Berger, M. Bunke and K. Sohr, "'An Android Security Case Study with Bauhaus," *Reverse Engineering (WCRE), 2011 18th Working Conference on*, pp. 179-183.

[24] A. Hense, F. Quadt and M. Romer, "'Towards a Mobile Workbench for Researchers," *e-Science, 2009. e-Science '09. Fifth IEEE International Conference on*, pp. 126-131.

[25] L.M. Murphy, "'The Busy Coder's Guide to Android Development ", ed. 1.3, 2008.

[26] F.J. DiMarzio, "'Android: A Programmer's Guide,", ed. 1, 2008.

[27] Google, "'Using the Emulator | Android Developers ", vol. 2014, no. 6/3/2014.

[28] Socketeq, "'Run Android on Windows -- Windroy, Android with Windows kernel ", vol. 2014, no. 6/3/2014.

[29] BlueStacks, "'BlueStacks ", vol. 2014, no. 6/3/2014.

[30] T.J. McCabe, "'A Complexity Measure," *Software Engineering, IEEE Transactions on*, vol. SE-2, no. 4, pp. 308-320.

[31] A. Smith, "'Smartphone Ownership - 2013 Update,".

[32] Cisco, "'Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013 – 2018,".

[33] Gartner, "'Gartner Says Worldwide Application Infrastructure and Middleware Market Revenue Grew 5.6 Percent in 2013 " *Gartner Announcements*, Thu, 15 May.

[34] Anonymous "'Data Protection Acts 1988 and 2003: Informal Consolidation - Data Protection Commissioner - Ireland ", vol. 2014, no. 6/10/2014.

[35] F.P.C. George, "'Driving Risks and Accidents,", vol. 2014, no. 6/6/2014.

[36] T. Åkerstedt, P. Philip, A. Capelli and G. Kecklund, "'Chapter 11 - Sleep loss and accidents—Work hours, life style, and sleep pathology ", vol. 2014, no. 6/6/2014.

[37] R.A. Barkley and D. Cox, "'A review of driving risks and impairments associated with attention-deficit/hyperactivity disorder and the effects of stimulant medication on driving performance " *J.Saf.Res.*, vol. 38, no. 1, pp. 113 <last_page> 128.

[38] L. Di Milia, M.H. Smolensky, G. Costa, H.D. Howarth, M.M. Ohayon and P. Philip, "'Demographic factors, fatigue, and driving accidents: An examination of the published literature " *Accid.Anal.Prev.*, vol. 43, no. 2, Mar, pp. 516-532.

[39] T. Manly, "'The Sustained Attention to Response Test (SART) - Neurobiology of Attention - Chapter 55 ", vol. 2014, no. 6/6/2014.

[40] J.R. Simon and A.M.J.r. Small, "'Processing auditory information: interference from an irrelevant cue," *Journal of Applied Psychology*, vol. 53, pp. 433.

[41] J.R. Stroop, "'
Studies of interference in serial verbal reactions," *Journal of Experimental Psychology*, vol. 18, pp. 643.

[42] S.W. Park, E.S. Choi, M.H. Lim, E.J. Kim, S.I. Hwang, K.I. Choi, H.C. Yoo, K.J. Lee and H.E. Jung, "'Association between unsafe driving performance and cognitive-perceptual dysfunction in older drivers " *PM R.*, vol. 3, no. 3, Mar, pp. 198-203.

[43] D. Romer, Y. Lee, C.C. McDonald and K.F. Winston, "'Adolescence, Attention Allocation, and Driving Safety " *Journal of Adolescent Health*, vol. 54, no. 5, pp. 6.

[44] C. Collet, C. Petit, A. Priez and A. Dittmar, "'Stroop color–word test, arousal, electrodermal activity and performance in a critical driving situation " *Biol.Psychol.*, vol. 69, no. 2, pp. 195 <last_page> 203.

[45] J.S. Bajaj, D.M. Heuman, R.K. Sterling, A.J. Sanyal, M. Siddiqui, S. Matherly, V. Luketic, R.T. Stravitz, M. Fuchs, L.R. Thacker, H. Gilles, M.B. White, A. Unser, J. Hovermale, E. Gavis, N.A. Noble and J.B. Wade, "'Validation of EncephalApp, Smartphone-based Stroop Test, for the Diagnosis of Covert Hepatic Encephalopathy " *Clinical Gastroenterology and Hepatology*.

[46] C.M. MacLeod, "'Half a century of research on the Stroop effect: an integrative review " *Psychol.Bull.*, vol. 109, no. 2, Mar, pp. 163-203.

[47] E.A. Dunne, "'Measuring Sustained Attention in a Driving Simulator,".

[48] S. Park, K.E. Hong, Y.H. Yang, J. Kang, E.J. Park, K. Ha, M. Park and H.J. Yoo, "'Neuropsychological and behavioral profiles in attention-deficit hyperactivity

disorder children of parents with a history of mood disorders: a pilot study " *Psychiatry.Investig.*, vol. 11, no. 1, Jan, pp. 65-75.

[49] V. Piai, A. Roelofs, D.J. Acheson and A. Takashima, "'Attention for speaking: domain-general control from the anterior cingulate cortex in spoken word production " *Front.Hum.Neurosci.*, vol. 7, Dec 9, pp. 832.

[50] G. Dong, X. Lin, H. Zhou and Q. Lu, "'Cognitive flexibility in internet addicts: fMRI... [Addict Behav. 2014] - PubMed - NCBI ", vol. 2014, no. 5/23/2014.

[51] M. Jackson, R. Croft, G. Kennedy, K. Owens and M. Howard, "'Cognitive components of simulated driving pe... [Accid Anal Prev. 2013] - PubMed - NCBI ", vol. 2014, no. 5/23/2014.

[52] S.T. Mueller and B.J. Piper, "'The Psychology Experiment Building Language (PEBL) and PEBL Test Battery " *J.Neurosci.Methods*, vol. 222, Jan 30, pp. 250-259.

[53] A.H. Al-Balbissi, "'Role of gender in road accidents " *Traffic Inj.Prev.*, vol. 4, no. 1, Mar, pp. 64-73.

[54] R. Factor, D. Mahalel and G. Yair, "'Inter-group differences in road-traffic crash involvement " *Accid.Anal.Prev.*, vol. 40, no. 6, Nov, pp. 2000-2007.

[55] M. Hasselberg, M. Vaez and L. Laflamme, "'Socioeconomic aspects of the circumstances and consequences of car crashes among young adults " *Soc.Sci.Med.*, vol. 60, no. 2, Jan, pp. 287-295.

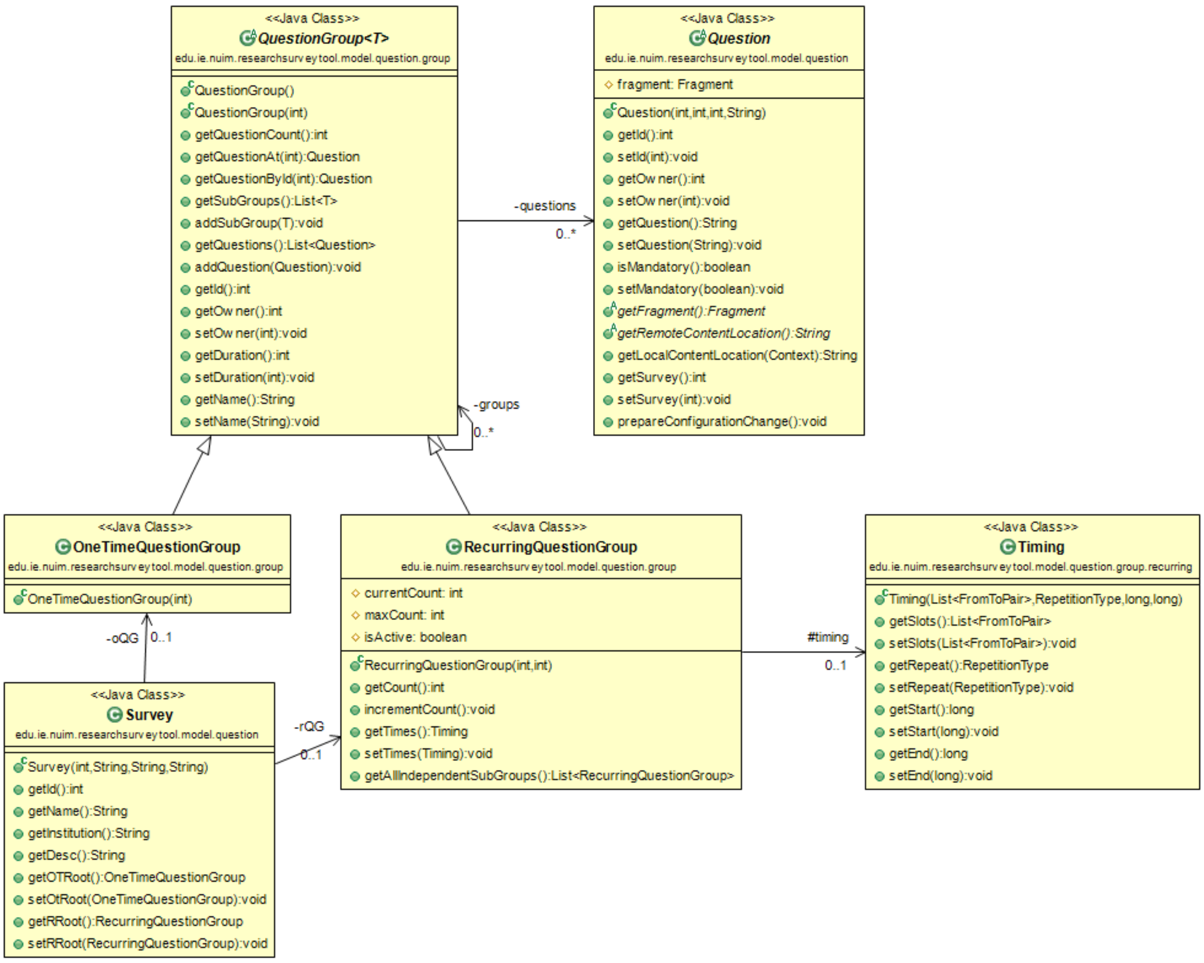
[56] S. Chandraratna, N. Stamatiadis and A. Stromberg, "'Crash involvement of drivers with multiple crashes " *Accid.Anal.Prev.*, vol. 38, no. 3, May, pp. 532-541.

[57] S. Brown, J. Timoney and T. Lysagth, "'Software Testing - Principles and Practice,", 2012.

[58] Yonghong Wu, Jianchao Luo and Lei Luo, "'Porting Mobile Web Application Engine to the Android Platform," *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pp. 2157-2161.

11 Appendices

11.1 A. Complete UML diagram of the data model


<<Java Class>>
QuestionGroup<T>
edu.ie.nuim.researchsurveytool.model.question.group
QuestionGroup()
QuestionGroup(int)
getQuestionCount():int
getQuestionAt(int):Question
getQuestionById(int):Question
getSubGroups():List<T>
addSubGroup(T):void
getQuestions():List<Question>
addQuestion(Question):void
getId():int
getOwner():int
setOwner(int):void
getDuration():int
setDuration(int):void
getName():String
setName(String):void
-questions
0..*
-groups
0..*
<<Java Class>>
Question
edu.ie.nuim.researchsurveytool.model.question
fragment: Fragment
Question(int,int,int,String)
getId():int
setId(int):void
getOwner():int
setOwner(int):void
getQuestion():String
setQuestion(String):void
isMandatory():boolean
setMandatory(boolean):void
getFragment():Fragment
getRemoteContentLocation():String
getLocalContentLocation(Context):String
getSurvey():int
setSurvey(int):void
prepareConfigurationChange():void
<<Java Class>>
OneTimeQuestionGroup
edu.ie.nuim.researchsurveytool.model.question.group
OneTimeQuestionGroup(int)
-oQG
0..1
<<Java Class>>
RecurringQuestionGroup
edu.ie.nuim.researchsurveytool.model.question.group
currentCount: int
maxCount: int
isActive: boolean
RecurringQuestionGroup(int,int)
getCount():int
incrementCount():void
getTimes():Timing
setTimes(Timing):void
getAllIndependentSubGroups():List<RecurringQuestionGroup>
#timing
0..1
<<Java Class>>
Timing
edu.ie.nuim.researchsurveytool.model.question.group.recurring
Timing(List<FromToPair>,RepetitionType,long,long)
getSlots():List<FromToPair>
setSlots(List<FromToPair>):void
getRepeat():RepetitionType
setRepeat(RepetitionType):void
getStart():long
setStart(long):void
getEnd():long
setEnd(long):void
-rQG
0..1
<<Java Class>>
Survey
edu.ie.nuim.researchsurveytool.model.question
Survey(int,String,String,String)
getId():int
getName():String
getInstitution():String
getDesc():String
getOTRoot():OneTimeQuestionGroup
setOtRoot(OneTimeQuestionGroup):void
getRRoot():RecurringQuestionGroup
setRRoot(RecurringQuestionGroup):void

11.2 A. Complete table of correlations between numeric responses, Stroop and driving measures

	age	license.year	self.score	cong.c	cong.inc	incong.c	incong.inc	neutral.c	neutral.inc	multiple.tries	rt.avg	rt.avg.c	rt.avg.c.cong	rt.avg.c.incong	rt.avg.c.neutral	stroop.score	dist.sd	dist.avg	user.acc	user.dec	user.br	rt.br.avg	missed.br.save
age	0																						
license.year	-0	0																					
self.score	-0	0.682590601	0																				
cong.c	-0	-0.05762063	-0.200454	0																			
cong.inc	0.5	0.229840032	0.071087	-0.225	0																		
incong.c	-0	0.189390752	0.211732	0.441	-0.3738	0																	
incong.inc	-0	0.052281052	0.363823	0.134	0.04374	0.37716	0																
neutral.c	-0	0.138377097	-0.023476	0.307	-0.4065	0.63717	-0.387154	0															
neutral.inc	-0	0.218562463	-0.28656	-0.096	-0.3101	0.29166	-0.458475	0.552982	0														
multiple.tries	0.5	0.09819473	-0.06368	-0.652	0.62017	-0.8102	-0.070535	-0.72365	-0.1666667	0													
rt.avg	0.4	-0.03453967	-0.025164	-0.822	0.45353	-0.8303	-0.161541	-0.68217	-0.1425032	0.940937536	0												
rt.avg.c	0.5	0.009916365	-0.012848	-0.786	0.50992	-0.8215	-0.095855	-0.72106	-0.167583	0.96943567	0.994	0											
rt.avg.c.cong	0.5	-0.0186611	-0.072932	-0.697	0.54042	-0.8693	-0.123808	-0.74618	-0.242824	0.978313514	0.975	0.9857	0										
rt.avg.c.incong	0.5	0.028655526	0.025204	-0.812	0.47296	-0.7342	0.005252	-0.69194	-0.0917299	0.935837838	0.962	0.9739	0.928681812	0									
rt.avg.c.neutral	0.4	0.00977456	0.009444	-0.771	0.51253	-0.8447	-0.120474	-0.73567	-0.2107495	0.961048356	0.993	0.996	0.988975746	0.95418836	0								
stroop.score	-0	0.067508853	-0.063375	0.856	-0.3843	0.79047	0.0860902	0.701215	0.2034232	-0.886716521	-0.99	-0.9723	-0.94021478	-0.951525795	-0.975040192	0							
dist.sd	0.4	0.291229884	0.21245	-0.664	0.58548	-0.7755	-0.211821	-0.58645	-0.2592255	0.885136528	0.875	0.8822	0.90621494	0.787708861	0.905574446	-0.84049185	0						
dist.avg	0.2	0.283553075	0.359344	-0.65	0.38175	-0.7766	-0.251165	-0.57726	-0.258664	0.74774348	0.82	0.8061	0.804402101	0.728112897	0.844362207	-0.82944802	0.919	0					
user.acc	0.3	0.223064014	0.212992	0.048	-0.0117	0.12699	-0.169922	0.430656	-0.1595785	-0.166669793	-0.18	-0.1985	-0.13527757	-0.295625931	-0.180748579	0.202010304	0.1428	0.01313	0				
user.dec	-0	-0.19298227	-0.225684	-0.008	0.07878	-0.127	0.107012	-0.37913	0.1805912	0.156003223	0.157	0.1727	0.113974545	0.265975859	0.156166246	-0.16157995	-0.147	-0.026	-0.9914	0			
user.br	0.3	0.159411369	0.234553	-0.032	-0.1449	0.12491	-0.041919	0.320798	-0.1986031	-0.142589461	-0.13	-0.1438	-0.09059183	-0.231580374	-0.128759112	0.118146256	0.1487	0.03853	0.96576	-0.99136	0		
rt.br.avg	-1	0.195172847	-0.18968	0.462	-0.0598	-0.0855	-0.420829	0.024382	0.1984212	-0.108016489	-0.19	-0.1919	-0.11511384	-0.305228006	-0.148363681	0.266706818	-0.029	0.03907	-0.2611	0.3082	-0.35	0	
missed.br.save	0.2	0.171048884	-0.4776	-0.172	0.31009	-0.4861	-0.282138	-0.31745	0.375	0.666666667	0.494	0.5345	0.575717393	0.499408636	0.50647728	-0.35990259	0.465	0.26374	-0.1815	0.19161	-0.198	0.32412	0