Code-Based In-Class Engagement App

Author: Andrew Graham

Supervisor: Manuel Maarek
Second Reader: Kenneth McLeod

A thesis submitted in fulfilment of the requirements for the degree of MSc Information Technology (Software Systems) in the School of Mathematical and Computer Sciences.

August 2018
Declaration of Authorship

I, Andrew Graham, declare that this thesis titled, ‘Code-Based In-Class Engagement App’ and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Andrew Graham

Dated: Thursday 16th August 2018
Abstract

A student response systems (SRS) is a system which allows a lecturer to pose questions to large cohorts of students. Typically, the results of these exchanges are displayed upon a whiteboard, along with the correct answers.

SRSs have been used successfully in the classroom to increase engagement and provide feedback to students and lecturers. This feedback guides student’s learning, highlighting their areas of weakness, and helps the lecturer keep track of the class’s state of knowledge.

Traditional SRS systems consist of multiple choice questions, which are by their nature easy to evaluate. The goal of this project is to produce a richer set of question types, to bring the benefits of SRSs to the teaching of programming.

Classes and methods can be tested against suites of unit tests, and unit tests can be tested against suites of mutated and errant code. Using these techniques, this project aims to develop and evaluate an SRS which uses these tests to provide immediate and quantifiable feedback to a large cohort of students.
Acknowledgements

I would like to thank my project-coordinator, Manuel Maarek, for his invaluable guidance and for suggesting the topic to begin with.
# Table of Contents

Declaration of Authorship ..................................................................................i  

Abstract ..............................................................................................................ii  

Acknowledgements .............................................................................................iii  

Table of Contents ...............................................................................................iv  

List of Figures ......................................................................................................ix  

List of Tables .......................................................................................................x  

Abbreviations ......................................................................................................xi  

Chapter 1 .............................................................................................................1  

Introduction .........................................................................................................1  

1.1 The Problem ..................................................................................................1  

1.2 Aims and Objectives ....................................................................................2  

1.2.1 Aims ........................................................................................................2  

1.2.2 Objectives ................................................................................................2  

1.3 Report Plan ...................................................................................................3  

Chapter 2 .............................................................................................................4  

Literature Review ................................................................................................4  

2.1 Teaching Programming Effectively ............................................................4  

2.1.1 Socratic and Didactic Tutoring Styles ....................................................4  

2.1.2 Classroom Engagement and Active Learning .......................................5  

2.1.3 Feedback ..................................................................................................7  

2.1.4 Gamification ............................................................................................8  

2.1.5 Current Practice .................................................................................... 8  

2.1.5.1 General ...............................................................................................8  

2.1.5.2 Software Testing and Debugging ......................................................11  

2.1.6 Summary ................................................................................................12  

2.2 Existing Student Response Systems and their Application .......................14  

2.2.1 Traditional Clickers ..............................................................................14  

2.2.1.1 Overview ..........................................................................................14  

2.2.1.2 Efficacy ..........................................................................................14  

2.2.1.3 Limitations ......................................................................................15  

2.2.2 Socrative ................................................................................................15  

2.2.2.1 Overview ..........................................................................................15  

2.2.2.2 Efficacy ..........................................................................................15  

2.2.2.3 Limitations ......................................................................................16  

2.2.3 Informa .....................................................................................................16  

2.2.3.1 Overview ..........................................................................................16  


Appendix B

Appendix A

References

Chapter 7

Recommendations for Further Work and Conclusions

6.1.1.2 Analysis.............................................................................................................49
6.1.2 User Evaluation ..................................................................................................49
6.1.2.1 Objectives and Hypotheses ...........................................................................50
  6.1.2.1.1 Engagement ...............................................................................................50
  6.1.2.1.2 Understanding ..........................................................................................50
  6.1.2.1.3 Feedback ...................................................................................................51
  6.1.2.1.4 Interaction ................................................................................................51
  6.1.2.1.5 Perceptions by Gender .............................................................................52
6.1.2.3 Usability Survey ..............................................................................................52
6.1.2.4 Further Questions ............................................................................................52
  6.1.2.4.1 Portability ..................................................................................................52
  6.1.2.4.2 Problems Encountered ............................................................................53
  6.1.2.4.3 Suggestions Received ...............................................................................53
  6.1.2.4.4 Liked Features ........................................................................................54
6.2 Conformance to Specification ..............................................................................54
  6.2.1 General Functional Requirements .................................................................54
  6.2.2 Student Functional Requirements .................................................................54
  6.2.3 Lecturer Functional Requirements ...............................................................55
  6.2.4 Non-Functional Requirements ......................................................................56

Chapter 7 ........................................................................................................................56

Recommendations for Further Work and Conclusions ..............................................56
  7.1 Recommendations for Further Work ..............................................................56
  7.2 Conclusions .........................................................................................................57

References .....................................................................................................................58

Appendix A .....................................................................................................................63

Lecturer’s Guide .............................................................................................................63
  A.1 Register and Login ...............................................................................................63
  A.2 Creating and Modifying a Lecture Plan .............................................................64
  A.3 Modifying a Problem ...........................................................................................65
  A.4 Moderating Lectures ..........................................................................................70
  A.5 Observing Lectures Presentations .....................................................................72

Appendix B .....................................................................................................................73

Student’s Guide .............................................................................................................73
  B.1 Joining a Lecture ..................................................................................................73
  B.2 Taking Part ..........................................................................................................73
  B.3 Asking a Question ................................................................................................75
  B.4 Completion – Seeing Your Results ....................................................................75
Appendix C ........................................................................................................................................... 77
Figures .................................................................................................................................................. 77
C.1 Connecting to a Presentation ........................................................................................................... 77
C.2 Sending a Solution for Evaluation ................................................................................................. 78
Appendix D ............................................................................................................................................ 79
User Survey .......................................................................................................................................... 79
D.1 The Consent Form ............................................................................................................................. 79
D.2 The Student Details Form ............................................................................................................... 79
D.3 The Place of Study Form ................................................................................................................. 80
D.4 The Past Study Form ....................................................................................................................... 80
D.5 The System Usability Scale (Brooke, 1996) ..................................................................................... 81
D.6 Objectives and Hypotheses Likert ................................................................................................. 82
D.7 Final Comments – Free Text Questions ......................................................................................... 82
Appendix E ............................................................................................................................................ 83
Evaluation Results ................................................................................................................................. 83
E.1 Load Test Results ............................................................................................................................. 83
E.2 Usability Survey Results ................................................................................................................... 84
E.3 Hypotheses and Objectives Evaluation Raw Results ...................................................................... 85
List of Figures

Figure 2.1: The interaction between various “constructs” which produce “learning performance”, (Blasco-Arcas et al., 2013) ......................................................................................................................... 6
Figure 2.2: Describing the “relationships between a number of issues relating to programming”, (Robins et al., 2003) ................................................................................................................................. 9
Figure 2.3: Feedback displayed to class, showing peer evaluation in the Informa SRS, (Hauswirth and Adamoli, 2013) ................................................................................................................................. 17
Figure 2.3: Feedback displayed to class, showing peer evaluation in the Informa SRS, (Hauswirth and Adamoli, 2013) ................................................................................................................................. 17
Figure 5.1: The Required Entities ..................................................................................................................... 32
Figure 5.2: The Iterations ................................................................................................................................. 40
Figure 5.3: Connecting to a Presentation (abridged) ......................................................................................... 42
Figure 5.4: Instructions and Solution Input ................................................................................................ 44
Figure 5.4: Instructions and Solution Input ................................................................................................ 44
Figure 5.5: Sending a Solution for Evaluation (abridged) ................................................................................ 45
Figure 5.6: The Student’s Results .................................................................................................................. 46
Figure 5.7: The Student’s Results .................................................................................................................. 46
Figure 5.8: The Cohort View .......................................................................................................................... 46
Figure 6.1: Submission Waves ......................................................................................................................... 48
Figure 6.2: Average Evaluation Time by Wave Chart .................................................................................... 49
Figure 6.3: Sys-Engage Combined Results Chart ......................................................................................... 50
Figure 6.4: Sys-Understand Combined Chart ............................................................................................. 50
Figure 6.5: Sys-Feedback Combined Results Chart ..................................................................................... 51
Figure 6.6: Sys-Interact Lecturer Combined Results Chart ................................................................ .......... 51
Figure 6.7: Sys-Interact Peers Combined Results ......................................................................................... 52
Figure A.1: The Lecturer Welcome Page ...................................................................................................... 63
Figure A.2: The lecturer registration page .................................................................................................... 63
Figure A.3: The Moderate Lectures page ....................................................................................................... 64
Figure A.4: The Edit Lecture Plan page ........................................................................................................ 65
Figure A.5: The Modify Problem Page .......................................................................................................... 66
Figure A.6: The Edit Tests and Groups page .................................................................................................. 67
Figure A.7: The Edit Test Group page ............................................................................................................ 68
Figure A.8: The Edit Test Group page ............................................................................................................ 69
Figure A.9: The completed problem .............................................................................................................. 70
Figure A.10: The Moderate Lectures page showing a newly spawned presentation .................................. 71
Figure A.11: A lecture in progress .................................................................................................................. 71
Figure A.12: The View Progress page ........................................................................................................... 72
Figure B.1: The Join Lecture page .................................................................................................................. 73
Figure B.2: Attempt Problem ....................................................................................................................... 74
Figure B.3: Lecture Summary Page .............................................................................................................. 76
Figure C.1: Connecting to a Presentation ....................................................................................................... 77
Figure C.2: Sending a Solution for Evaluation ............................................................................................ 78
Figure D1: The System Usability Scale, (Brooke, 1996) .................................................................................. 81
List of Tables

Table 3.1: General Functional Requirements ................................................................. 23
Table 3.2: Student Functional Requirements ................................................................. 23
Table 3.3: Lecturer Functional Requirements ............................................................... 24
Table 3.4: Non-Functional Requirements ....................................................................... 25
Table 6.1: Average Evaluation Time by Wave .................................................................. 49
Table 6.2: Sys-Engage Combined Results Table ............................................................. 50
Table 6.3: Sys-Understand Combined Results ................................................................. 50
Table 6.4: Sys-Feedback Combined Results .................................................................... 51
Table 6.5: Sys-Interact Lecturer Combined Results ......................................................... 51
Table 6.6: Sys-Interact Lecturer Combined Results ......................................................... 52
Table 6.7: Platform Used ................................................................................................. 52
Table 6.8: General Functional Requirements Implementation ........................................ 54
Table 6.9: Student Functional Requirements Implementation ........................................ 54
Table 6.10: Lecturer Functional Requirements ............................................................... 55
Table 6.11: Non-Functional Requirements ...................................................................... 56
Table D.1: Consent Form ................................................................................................. 79
Table D.2: Student Details Form ..................................................................................... 79
Table D.3: Place of Study Form ...................................................................................... 80
Table D.4: The Past Study Form .................................................................................... 80
Table D.5: Objectives and Hypotheses Likert ................................................................ 82
Table D.6: Final Comments – Free Text Questions ......................................................... 82
Table E.1: Load Test Results .......................................................................................... 83
Table E.2: Usability Survey Results ................................................................................ 84
Table E.3: Hypotheses and Objectives Evaluation Raw Results ..................................... 85
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise Java Bean</td>
</tr>
<tr>
<td>Java EE</td>
<td>Java Enterprise Edition</td>
</tr>
<tr>
<td>JSP</td>
<td>JavaServer Pages</td>
</tr>
<tr>
<td>JPA</td>
<td>Java Persistence Architecture</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>PBL</td>
<td>Problem-Based Learning</td>
</tr>
<tr>
<td>SRS</td>
<td>Student Response System</td>
</tr>
<tr>
<td>STEM</td>
<td>Science Technology Engineering and Mathematics</td>
</tr>
<tr>
<td>SUS</td>
<td>System Usability Scale</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification and Validation</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This section discusses the problem which the project addresses in Section 1.1 The Problem, the project’s aims in Section 1.2.1 Aims, followed by the objectives, including the software tool to be produced and the questions which it seeks to address in Section 1.2.2 Objectives.

1.1 The Problem

A student response systems (SRS) can be used to increase classroom engagement (Blasco-Arcas et al., 2013); however, these systems are generally best suited to multiple choice questions and are unable to automatically evaluate arbitrary solutions to programming problems.

Increased classroom engagement leads to better pedagogical outcomes (Ahlfeldt et al., 2005; Blasco-Arcas et al., 2013; Carini et al., 2006) and interaction amongst peers and between students and their teachers augments this process (Blasco-Arcas et al., 2013). The use of Socratic tutoring, where the student’s reasoning process is drawn out by means of subtle pointers rather than explicit information, can be beneficial (Rose et al., 2001).

The teaching of programming could benefit from the increased engagement afforded by SRSs; however, there currently exists no satisfactory solution for the immediate evaluation and provision of feedback in this domain.

Software testing is a generally neglected skill in programming education, but it is essential for the Verification and Validation (V&V) of software systems (Carrington, 1996). It would also be useful, if an SRS could be extended to address this need.

Java is the most used programming language in higher education (Michaelson, 2015); therefore, targeting this language would have the most impact.

This project will explore a potential solution: the implementation of an SRS optimised for the teaching of programming. This system shall allow lecturers to develop and present programming problems in lectures, as well as providing immediate evaluation and feedback, tailored to the user, with lecturers receiving a detailed cohort-wide view, with individual scores, and students receiving feedback on their own performance, along with summaries of their peers’ progress.

The evaluation of solutions will be provided by executing each solution against a suite of unit tests, or, if testing unit tests, a selection of errant programs. JUnit will be integral to this evaluation and feedback.
Chapter 1 Introduction

This project will ultimately address the overarching research question and the hypotheses into which it is decomposed, introduced in Section 1.2.2 Objectives. These will be evaluated by means of a mock lecture and survey.

1.2 Aims and Objectives

1.2.1 Aims

This project aims to implement a novel SRS solution, allowing lecturers to deliver programming problems to large cohorts of students whilst providing immediate and useful feedback; and to evaluate the efficacy of this solution.

1.2.2 Objectives

The objectives of the project are set out below; the requirements for their fulfilment are presented in Sections 3.1.3 Functional Requirements and 3.1.4 Non-Functional Requirements. A detailed account of their implementation is provided in Chapter 4 Design and Implementation.

The project objectives are as follows:

<table>
<thead>
<tr>
<th>Objective</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Augmentation</td>
<td>To provide a tool to augment traditional programming lectures</td>
</tr>
<tr>
<td>Engagement</td>
<td>This tool must engage students in practical programming exercises</td>
</tr>
<tr>
<td>Interaction</td>
<td>These exercises should encourage student interaction – between peers and with their lecturer</td>
</tr>
<tr>
<td>Feedback</td>
<td>To provide a means for immediate feedback for students, grading their solutions and showing them how they compare with the rest of their cohort</td>
</tr>
<tr>
<td>Progress</td>
<td>To provide a means for immediate feedback for lecturers, allowing them to monitor the state of the class’s knowledge via a breakdown of the cohort’s results across a series of problems</td>
</tr>
<tr>
<td>Java</td>
<td>This tool must target the Java programming language</td>
</tr>
<tr>
<td>Anonymity</td>
<td>To provide a channel for the anonymous asking of questions</td>
</tr>
<tr>
<td>Portability</td>
<td>This tool must be available on the widest range of devices practically possible</td>
</tr>
</tbody>
</table>

This project shall try to answer the following question: Can a student response system, which is optimised for the teaching of programming, improve learning outcomes in a programming lecture?

To answer this, I shall implement a new system and use it to test the following hypotheses:

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engagement</td>
<td>A student response system optimised for teaching programming will yield higher student engagement than the use of a general one.</td>
</tr>
<tr>
<td>Understanding</td>
<td>A student response system optimised for teaching programming will provide the student with a better understanding of new programming concepts than the use of a general one.</td>
</tr>
<tr>
<td>Feedback Student</td>
<td>A student response system optimised for teaching programming will provide the student with useful feedback, highlighting their areas of weakness.</td>
</tr>
<tr>
<td>Feedback Lecturer</td>
<td>A student response system optimised for teaching programming will provide the lecturer with useful feedback, allowing them to keep track of the state of the class’s knowledge.</td>
</tr>
</tbody>
</table>
1.3 Report Plan

This report explores the issues in programming education which the project aims to address and how to address them; Chapter 2 Literature Review discusses approaches to improving programming education, deficits in current practice and recommendations for further activity, based on the review.

Chapter 3 Methodology discusses how the recommendations from Chapter 2 are to be implemented, how the objectives discussed above are to be achieved, and how the success of the project and the hypotheses are to be evaluated.

Chapter 4 Professional, Legal Ethical and Social Issues sets out the ethical and legal constraints in which the system must operate.

Chapter 5 Design and Implementation discusses the design and technical aspects of developing the new system.

Chapter 6 Results presents the results of the evaluations and analysis, as well as the implementation status of all of the requirements.

Chapter 7 Recommendations for Further Work and Conclusions introduces recommendations for further work based on the results of the project, followed by conclusions, reflecting on the what has been achieved and summarising what has been learnt.
Chapter 2

Literature Review

Many different approaches to teaching programming have been attempted and evaluated. Some general pedagogical concepts and how they can be used for the effective teaching of programming, as well as current practice in programming education, some novel solutions being attempted, and their strengths and weaknesses are discussed in Section 2.1 Teaching Programming.

Much literature pertaining to classroom engagement, how it can be increased, and its effects on learning outcomes exists. Section 2.1.2 Classroom Engagement and Active Learning introduces some of this literature, as well as the concept of an SRS system and its use in testing a model of the relationships between engagement, interaction and learning performance.

Feedback has been well documented as a means of improving learning outcomes. Research discussing the importance of feedback is introduced in Section 2.1.3 Feedback.

Gamification has been used successfully as a means of providing feedback and boosting engagement. This is explored in detail in Section 2.1.4 Gamification.

Much literature exists discussing programming education as it is currently practiced. This is explored in detail, covering successful approaches as well as shortcomings in standard courses, in Section 2.1.5 Current Practice.

To boost classroom engagement and improve learning outcomes, many SRS systems have already been developed. In Section 2.2 Existing Student Response Systems, some notable implementations are explored, along with their key features and efficacy.

Recommendations for further activity, based on the above, are suggested in Section 2.3 Recommendations for Further Activity.

2.1 Teaching Programming Effectively

2.1.1 Socratic and Didactic Tutoring Styles

Rose et al. (2001) describe the Socratic tutoring style as being “characterized by an emphasis on eliciting information from students through a directed line of reasoning”. They compare this with a didactic tutoring style, “in the Didactic tutoring style, the tutor begins extended interactions with students by presenting them with an explanation of the material the student is meant to learn in the interaction”.
Their comparisons suggest the following: didactic instruction focuses on the presentation of complete information, with feedback consisting of a restating of facts, in contrast to a Socratic approach which encourages the students to infer facts for themselves, with feedback consisting of “leading questions”, guiding the student to discover the answer.

They conducted a survey, splitting students into two groups: those in the “didactic condition” received feedback in the form of reiteration of previous theory; those in the “Socratic condition” received question-based feedback, which “explained as little as possible”. They found that the students who received Socratic tutoring learned more effectively.

A key component of this “Socratic condition” could be realised in an SRS: the drawing out of the students reasoning using hints and pointers.

2.1.2 Classroom Engagement and Active Learning
Carini et al. (2006) studied the relationships between different types of engagement and performance, the effect of certain student characteristics on performance and how the link varied between institutions. They found a positive correlation between higher engagement and better learning outcomes, backing up the findings of previous research. This correlation varied between institutions, suggesting that other factors could be involved.

According to Fee and Holland-Minkley (2010), Problem-Based Learning (PBL) provides a structure in which students apply a course’s concepts together to solve a problem in a context relevant to the course. They describe a process that begins with well-structured problems, which become less structured as students develop their skills, with mentoring provided by tutors. They add that the goal of these increasingly broad problems is that students will be able to go on to solve unfamiliar problems for themselves in the real world.

The problems described by Fee and Holland-Minkley (2010) involve large projects, often completed in groups. They describe a process where, as the programme progresses the problems posed become more loosely defined, eventually getting the students to define what the product should be, within broadly defined constraints.

Surveying classroom engagement and PBL, Ahlfeldt et al. (2005) found that, “Students participate more in a classroom and also report a better understanding of course concepts when steps are taken to actively engage them.” Testing the use of PBL, they found that students taught under it reported higher engagement. They describe PBL as “an innovative teaching method for engaging students in solving practical and relevant problems”. This research was conducted with Physics students; whilst this is not programming, as a Science Technology Engineering and Mathematics (STEM) subject, the results can be partially generalised to Computer Science and programming.
Whilst PBL, as implemented by Fee and Holland-Minkley (2010), is too long and complex to implement in short SRS sessions, the solving of realistic programming problems is achievable, and could bring the benefits of improved engagement and understanding described above.

Prince (2004) reviewed the literature pertaining to active learning. He suggests that breaking up lectures with activities provides opportunities for students whose minds have wandered to reengage and states that student engagement is widely accepted as important. He adds, “Introducing activity into lectures can significantly improve recall of information”. He states further that collaborative learning promotes higher academic achievement and knowledge retention.

Chickering and Gamson (1987) enumerate “Seven Principles for Good Practice in Undergraduate Education”. They assert that, “Learning is not a spectator sport.” They state that students learn more by applying what they learn, rather than just trying to memorise it, and suggest “structured exercises, challenging discussions, team projects, and peer critiques” as practical examples.

Blasco-Arcas et al. (2013) published a study of SRS clickers, covering their role in classroom engagement. They developed a model detailing the relationships between various “constructs” and their contribution to “learning performance”. Figure 2.1 (Blasco-Arcas et al., 2013), illustrates these relationships.

![Figure 2.1: The interaction between various “constructs” which produce “learning performance”, (Blasco-Arcas et al., 2013).](image)

Defining “active learning”, Blasco-Arcas et al. (2013) state, “[Active learning] refers to practices that engage students in the learning process and includes techniques where students do more than passively listen to a lecture”.

They tested several hypotheses in a sample of 198 undergraduate business students. The teachers involved taught identical material and had the same level of experience with SRSs. The students completed 10 sessions, consisting of 7 multiple-choice questions, followed by a final test covering all the materials. In total these questions made up 5% of the student’s overall mark, so it is important to consider that this may yield different results than purely formative sessions, because in this case, producing the correct results may appear more important. Groups of students were also prompted to generate some of their own questions.
Blasco-Arcas et al. (2013) gave the participants a 7-point Likert questionnaire. The goal of the survey was not specifically to test the efficacy of SRSs; therefore, they did not test learning outcomes between programmes with and without them. The purpose was to provide a “conceptual framework” describing a means by which they may be effective, and to provide data to support that. The results would, however, be more illuminating if they gave similar questions to a none-SRS group on the same course and compared outcomes. They suggest that the results may have been affected by a “novelty effect”, since the clickers were new to the course, and that it is not a given that these results can be generalised across subjects.

Their results back up the model which they propose; they state that active collaborative learning and engagement improve learning outcomes, and these are promoted by student-to-student interaction and student-to-teacher interaction. They conclude that by promoting this interaction, student engagement and active collaborative learning, “clickers are shown to be a powerful technological tool to enhance student learning performance”.

Michael Cavanagh (2011) conducted a survey of 113 trainee mathematics teachers, applying the concepts of cooperative learning, thus creating interaction between students, to boost engagement in a study comprising of one semester of “lectorials”. These lectorials were designed to create discussion amongst students and create a deeper understanding of the subject matter. The students were given a questionnaire, of which 94 were analysed as these came from students who had attended at least 11 of the 13 lectures.

Michael Cavanagh (2011) found that 91 of 94 students said that the cooperative learning activities helped them understand the course contents, with the common reasoning that, “the lectorials provided numerous opportunities for students to become active participants in their learning”. Students felt that active involvement was promoted by the discussions and sharing views improved their understanding.

All the students found that the activities helped them maintain interest and remain attentive, with the “diversity of activities” helping pass time. These practical activities were found to improve students’ confidence in the subject.

Michael Cavanagh (2011) also found that students like lectorials with a “clear focus on one or two central ideas” and activities which answer questions related to those ideas in a coherent way.

2.1.3 Feedback

In their paper, “Seven Principles for Good Practice in Undergraduate Education”, Chickering and Gamson (1987) state the importance of prompt feedback. Asserting that, “Knowing what you know and don't know focuses learning.” They expand that students need frequent feedback, so they know where to improve and that, “feedback is central to learning and improving performance”.


Feedback is essential to learning, with more frequent feedback yielding higher efficiency learning (Kapp, 2012). As well as engaging the learner, feedback helps learners identify their weaknesses; Lahtinen et al. (2005), discussed above, identified students’ lack of awareness of their own limitations as impeding their learning of programming skills.

2.1.4 Gamification

Kapp (2012) explains that gamification engages players by providing direct feedback with direct consequences to their actions. He states that a game should contain quantifiable feedback and, to keep learners’ attention, should engage them in a high level of activity. Kapp (2012) offers the following definition of gamification:

“Gamification is using game-based mechanics, aesthetics, and game thinking to engage people, motivate action, promote learning, and solve problems.”

Kapp (2012) views “freedom to fail” as an important concept in gamification, so students can experiment with wrong answers and their consequences in a safe environment. He states that games afford this through multiple opportunities to attempt tasks and multiple means of success. He adds that upon giving wrong answers, students should be made to determine the correct answer for themselves, and receive feedback on the consequences of their decisions, and that learning from failure should be encouraged. This could be afforded to the student by allowing multiple attempts to submit a solution, providing feedback each time.

Kapp (2012), in his article, introduces the concept of an interest curve. He states that this should begin with a high interest event, to grab the students’ attention, with periodic interesting events to maintain their engagement, followed by a final high interest event at the end, so the game ends with the student holding a residual interest in the topic. These events could be mapped to an SRS session. Commencing with the first problem could be a high-interest event, with the feedback received from each successive problem as medium-interest events and the final result and ranking being a high-interest event.

2.1.5 Current Practice

2.1.5.1 General

In this section, I look at the current state of programming education and some of the novel approaches taken to try to improve it.

Robins et al. (2003) present a paper discussing techniques for learning and teaching object-oriented and procedural programming effectively, reviewing the current literature of the time. They found a gap between the learning of basic program elements and how to apply them to real problems. They suggest the possibility of teaching usage patterns or teaching elements, such as a for loop, in the context of real problems.
They note that the ability to comprehend existing code does not necessarily correspond to the ability to generate code, so both skills need to be practiced. They dispute the idea that OOP is easier to understand than other paradigms and state that, “it may be necessary, particularly for weaker students, to devote particular attention to procedural concepts, flow of control, flow of data, and design”, even in OOP languages. They suggest that students need to learn: how to model the program domain; tracking and debugging skills, and the language’s “notional machine”. They state that, “Several authors have suggested, however, that the most important deficits relate to the underlying issues of problem solving, design, and expressing a solution/design as an actual program.” They add that practical exercises are an important means of addressing this and that these can be further augmented by use of an explicit software development methodology.

Figure 2.2 (Robins et al., 2003), shows the issues related to learning programming, as described in their paper, listed by stage.

<table>
<thead>
<tr>
<th>Design</th>
<th>Knowledge</th>
<th>Strategies</th>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>of planning methods, algorithm design, formal methods</td>
<td>for planning, problem solving, designing algorithms</td>
<td>of problem domain, notional machine</td>
<td></td>
</tr>
<tr>
<td>Generation</td>
<td>of language, libraries, environment / tools</td>
<td>for implementing algorithms, coding, accessing knowledge</td>
<td>of desired program</td>
</tr>
<tr>
<td>Evaluation</td>
<td>of debugging tools and methods</td>
<td>for testing, debugging, tracking / tracing, repair</td>
<td>of actual program</td>
</tr>
</tbody>
</table>

Figure 2.2: Describing the “relationships between a number of issues relating to programming”.

(Robins et al., 2003).

Any teaching solution this project produces should address all these issues, or at least have scope to be extended to. Robins et al. (2003), further suggest that programming strategies should receive greater attention.

Lahtinen et al. (2005) studied common difficulties encountered by novice programmers. Based on the results of an international survey of 500 students, they offer recommendations for the design of new approaches to the subject. They found that the most difficult issues in understanding programming were: designing solutions targeted at specific problems, splitting up programs into units of functionality, and debugging. In their summary discussion, they state, “the biggest problem of novice programmers does not seem to be the understanding of basic concepts but rather learning to apply them”.

9
Chapter 2 Literature Review

They also found that learning situations which involved practical exercises were rated as more useful by the students and example programs were found to be the most useful type of material by teachers and students; they state, “The more practical and concrete the learning situations and materials are, the more learning takes place.” Further, they posit that students may overestimate their own understanding and fail to recognise their difficulties; this would seem to point to a need for more feedback.

Winslow (1996) produced a paper, also cited in the papers discussed above (Lahtinen et al., 2005; Robins et al., 2003), discussing the psychology of “Programming Pedagogy”. Winslow (1996) refers to “[a] large number of studies concluding that novice programmers know the syntax and semantics of individual statements but they do not know how to combine these features into valid programs.” And expands, “[l]earning the concepts and techniques of a new language requires writing programs in that language. Studying the syntax and semantics is not sufficient to understand and properly apply the new language.” More specifically they add, “we begin by combining the new information with general problem-solving techniques and advance to generating new, task-specific problem-solving procedures”. Winslow (1996) breaks problem solving in programming down to the following steps:

"1. understand the problem,
2. determine how to solve the problem:
   a. in some form and
   b. in computer compatible form
   (note that novices have trouble going
    from a to b),
3. translate the solution into computer language
   program, and
4. test and debug the program."

As with the issues highlighted by Robins et al. (2003), all the steps listed by (Winslow, 1996), should be addressed by any teaching solution, or be capable of being added. They further argue that, “A final step in problem solving is to consciously analyze what has been done along with how and why it has been done. All students seem to need continual reminding of this step.”

Winslow (1996) highlights the disconnect between being able to generate a program and being able to comprehend one, stating that both skills must be taught. He argues that “most texts” skip or give only “cursory coverage” to debugging and this skill depends on the student’s ability to comprehend programs, not their ability to generate them, and that they require a strategy to do this.

Winslow (1996) ends with the statement:
Chapter 2 Literature Review

“Psychological studies of expertise in general and computer programming expertise in particular show that turning a novice into an expert is impossible in a four year program. Competence, however, is possible. Reaching this level requires mastering basic facts, features and rules and being able to consciously plan and carry through a problem solution in specified areas.”

This suggests that facilitating the goal of reaching competence would be a sensible target for any system developed.

2.1.5.2 Software Testing and Debugging
Carrington (1996) differentiates between testing and debugging, stating that testing identifies the bugs, and debugging, their causes. He divides testing into three phases:

1. Planning: the identification of test cases, consisting of their input and expected output.
2. Execution: the execution of the tests, supplying the specified inputs and recording the outputs.
3. Analysis: comparing the expected results with the actual results.

Carrington (1996) splits test cases into functional and structural. He describes functional testing as black box testing, with the emphasis on inputs and outputs; in this case the function itself is like a black box whose inner workings are invisible to the user. He describes structural testing as white box testing, where the goal is to test as many paths within the function as possible.

Carrington (1996), describes testing as “relatively neglected”, due to its lack of “glamour” in comparison to other aspects of software development, whilst emphasising its importance as a means of verification and validation (V&V). He goes on to add that testing requires practical experience. He highlights a conflict of interest, wherein students avoid finding bugs in their own program, as they may see this as personal failure, finding it more interesting to test those of their teachers and peers. He proposes using exercises based upon other’s code as a solution.

Shepard et al. (2001)’s paper, “More Testing Should Be Taught,” articulates this point, based upon the assertion that 50% of development resources are expended upon this task. Whilst they emphasise the importance of teaching general V&V techniques, they state the importance of testing to evaluate the product, adding, “It is important to teach an appreciation and enthusiasm for this wide range of activities, to try to change the prevailing mindset of testing as a necessary evil.” They concentrate on the importance of V&V before the final testing phase, to reduce the number of errors discovered in the final testing phase, where fixing them is most expensive.

Winslow (1996), discussed above, also states that programming is undertaught and that more emphasis should be placed upon it.

Fraser (2017), discussing the gamification of software testing, states that it is highly neglected in programming education and lags behind in research, with software bugs leading to economic damage.
and even fatalities. He suggests gamification as a potential solution for teaching this complicated and unengaging task, introducing the elements of fun and competition, to reengage the student and elicit better learning outcomes.

Seela (2018) describes mutation testing as a process in which unit tests themselves can be tested. According to her article, this is a process in which many altered pieces of code are generated to produce errors, with the good unit tests catching all the errors. This process is the inverse of unit testing, with errant mutants providing tests for unit tests. This technique could be used in an SRS to evaluate unit tests.

2.1.6 Summary

PBL increases engagement (Ahlfeldt et al., 2005). This methodology involves projects in which students, often working in groups, attempt to solve relevant problems, defining their own outcomes within broad constraints, to apply and consolidate the course contents, whilst developing their problem-solving skills (Fee and Holland-Minkley, 2010).

Since the scope of this project is to augment current lectures and programmes, not replace them, the methodology as described is too large and time consuming. It has, however, seen positive benefits through increased engagement; therefore, this project shall take a key component of this methodology, realistic practical problems, and produce exercises based upon them.

Drawing out the students’ reasoning, teaching in a Socratic manner, leads to better outcomes (Rose et al., 2001). Based on this observation, it would seem desirable to avoid providing too much information to the student; multiple-choice questions reveal a lot of information, displaying a correct solution amongst the options.

Practical exercises are important to improve problem solving skills when learning to program (Lahtinen et al., 2005, Robins et al., 2003). With these points in mind, moving the emphasis away from multiple choice questions, and to more free-form practical questions would be productive.

The ability to generate code and the ability to comprehend it are separate skills that do not necessarily correlate, so both need to be practised (Robins et al., 2003; Winslow, 1996).

Reaching expertise in programming within the duration of a standard undergraduate course is impossible (Winslow, 1996); therefore, competence is a more realistic goal.

Higher classroom engagement is associated with better learning (Ahlfeldt et al., 2005; Blasco-Arcas et al., 2013; Carini et al., 2006). There is a positive correlation between engagement and academic performance (Carini et al., 2006). Tasks that seem relevant and practical in real-life, as well as varied activities increase student engagement (Michael Cavanagh, 2011). Active learning boosts recall and collaborative learning boosts both knowledge retention and academic performance (Prince, 2004).
Interaction amongst peers and between students and their tutor boosts engagement and active collaborative learning, and through these, their learning outcomes (Blasco-Arcas et al., 2013).

Higher-frequency feedback leads to higher-efficiency learning (Kapp, 2012). Students are often held back by a lack of knowledge of their own limitations (Lahtinen et al., 2005). Feedback can highlight these gaps in knowledge (Chickering and Gamson, 1987; Lahtinen et al., 2005).

Some elements of gamification could be applied to the problem. Quantifiable feedback and freedom to fail, with students being allowed to learn from their mistakes and correct them, help learning (Kapp, 2012). To maintain engagement, the task should begin with a high interest event, punctuated by moderate interest events, ending in a high interest event (Kapp, 2012). This engagement promotes learning (Carini et al., 2006; Blasco-Arcas et al., 2013).

An initial high interest event could come in the form of the initial problem, followed by moderate interest events in the form individual problem scores and a final high-interest event could come in the form of the final score and ranking. Scores could be anonymised and ranked to give an element of competition, without creating embarrassment for the lower scorers.

Freedom to fail could be afforded to the student by allowing multiple attempts to submit a solution, providing feedback each time.

The ability to comprehend code, required to test and debug it, is a weakness amongst many students (Lahtinen et al., 2005). Many existing texts give insufficient coverage to the skills required for the testing and debugging of code (Winslow, 1996). Testing is an area which receives too little coverage in education (Carrington, 1996; Fraser, 2017; Shepard et al., 2001; Winslow, 1996).

Students are often better at finding bugs in tutor and peer code, as finding bugs in their own can be perceived as a personal failure (Carrington, 1996).

Black-box testing describes tests which focus on the inputs and outputs of a function, whereas white-box testing is concerned with the implementation, trying to cover as many paths of execution as possible (Carrington, 1996).

We can carry out black-box testing with unit tests. Unit tests themselves can be tested by creating mutants, mutated versions of the code-under-test, useful ones of which produce errors, attempting to defeat the unit tests (Seela, 2018).

Providing problems based on black-box testing in an SRS is a more obtainable goal than debugging or white-box testing, as unit tests can be easily evaluated with mutants. A new teaching solution should provide problems in which students generate solutions to problems posed by a lecturer; these solutions can be evaluated with unit tests. An inverse problem should exist where students provide unit tests, evaluated using mutation testing.
2.2 Existing Student Response Systems and their Application

One possible means of implementing the core principal of the Socratic tutoring style, drawing out the student’s reasoning, with a large cohort of students, is the use of SRSs. An SRS could provide feedback to students and hopefully help them recognise their own difficulties early on, dealing with the issue posited by Lahtinen et al., (2005), that students overestimate their own abilities and fail to recognise their difficulties. Given a rich enough answer format, the student could be prompted to solve practical problems, hopefully helping to bridge the gap between knowledge of a programming language’s elements and that of combining them into an effective solution, as identified in the papers reviewed previously (Lahtinen et al., 2005; Robins et al., 2003; Winslow, 1996). Lahtinen et al. (2005) explicitly states the need for practical exercises. Requiring student interaction, these systems should theoretically improve student engagement, and therefore learning and outcomes. Winslow (1996) states the importance of students analysing their work after completing a problem-solving task; a well-structured SRS session could accommodate this. An SRS could also implement many of the elements of gamification discussed by Kapp (2012).

2.2.1 Traditional Clickers

2.2.1.1 Overview

The first generation of SRSs used specialised handheld devices called “clickers”, costing each student up to $60USD, to take responses from the class, before displaying them in histograms embedded in PowerPoint slides (Taylor, 2013).

Thomas et al. (2015) state that, “the system allows students to respond anonymously, which encourages students to participate who might not otherwise out of fear of public embarrassment or failure”.

The use of an electronic SRS system is much more scalable than any manual polling technique; Thomas et al. (2015) point out that, “relying on students to raise their hands to respond to questions in a lecture format limits responses to a single student at a time”. This combined with anonymous responses, allowing students to overcome their apprehension, affords the opportunity to increase the number of students participating in an exchange by a large degree.

2.2.1.2 Efficacy

Thomas et al. (2015) studied the effect of using a traditional SRS system, using “clickers” to answer questions in PowerPoint slides. They found in their study, no improvement or worsening in knowledge acquisition using the clickers, even though they had higher levels of self-reported engagement. Studying knowledge attainment and engagement in an SRS group, a written response group and a group providing no responses, they found that the group who provided no responses, had the lowest self-reported engagement but attained as much knowledge as both the written and SRS response groups.
This highlights an important point: higher engagement is only of value if that engagement is in a useful activity.

Thomas et al. (2015) concede that findings in studies such as these in the social sciences are often difficult to generalise, with this study concentrating on students in one narrow area of one field. They also concede that the presentation of the material may have influenced the students’ ability to retain the information: the lecturer did not restructure the lectures with the addition of the SRS in mind; and their study was affected by the learning curve of the technology, with technological failure causing the loss of all data from the first semester.

This suggests, that to be effective, SRS activities must be useful. Better activities, well integrated into lectures, may yield better results than those of Thomas et al. (2015). It is also important that lecturers retain access to data, so they can use feedback to address the class’s weaknesses.

Blasco-Arcas et al. (2013), discussed in Section 2.1.2 Classroom Engagement and Active Learning, found clickers to increase student engagement. More data on actual attainment, with and without clickers, would be instructive.

2.2.1.3 Limitations

Traditional clicker systems often come with only a numeric keypad, making the provision of questions with rich content difficult; these systems are best suited to multiple choice answers. Many of these systems require the installation of specialised infrastructure and proprietary systems can lack flexibility and often suffer from a high unit cost.

2.2.2 Socrative

2.2.2.1 Overview

Socrative allows the creation of quizzes and “quick questions” in the form of multiple choice; true or false; and short answers. Visualisations and reports of results can be generated, and students can connect via a wide range of internet connected devices (MasteryConnect, 2018). These devices are generally the students’ own smartphones or laptops. This frees them from having to buy specialist hardware, and, since it uses the internet, frees universities from having to installing new infrastructure.

2.2.2.2 Efficacy

Guarascio et al. (2017) state that the backchannel provided by Socrative, through which students can ask questions anonymously, can engage students and create interaction with the tutor, by affording them the opportunity to ask questions which they may otherwise be too nervous to ask.

Dakka (2015) conducted a study into the efficacy of Socrative. The results show an improvement in marks where Socrative is used. A qualitative analysis, in the form of a survey, was also conducted. In
the survey, Dakka (2015) found, “that the students felt that they are actively collaborating in their learning experience, have the freedom to participate in their learning experience, improved their understanding of material, improved their learning experience, and enhanced the exchange of the information with the lecturer”.

The lectures in this survey were designed specifically to make use of Socrative; some research, such as that on traditional clickers, had the questions interleaved into a standard lecture. The use of students on an engineering course reflects content that is closer to object-oriented programming than other surveys, but it is still very different.

2.2.2.3 Limitations
Socrative affords a wide range of question types and supports many platforms, so its limitations are mainly domain specific. Lacking support for compile time testing and aggregation of answers to programming questions, answers to these questions supplied as plain text cannot be automatically evaluated.

2.2.3 Informa
2.2.3.1 Overview
Hauswirth and Adamoli (2013) conducted a study in which they developed a software SRS system, the Informa platform, with the specific purpose of teaching programming. They explain that this “allow[s] for much richer problem types than the traditional multiple-choice question”, affording “students a much higher degree of freedom in solving a problem, and thus more opportunities for making mistakes”, complimenting “a pedagogical approach that allows students to learn from mistakes of their peers”. They call this methodology “Solve & Evaluate”.

According to their report, Informa supports a variety of question types using plugins; these plugins can provide their own visualisations for questions whose results cannot be summed up in a histogram. They break down the problem of teaching programming skills into four key areas: syntax, types, control flow and coding, with the skill of coding being the aggregate of the first three.

They explain the general pattern of the questioning as: have the students solve the problem, discuss the problem, then reveal the answer. They approached the problem of varying task completion times by presenting students who had completed the tasks quickly with solutions provided by their peers, who then used this potential down-time to evaluate these solutions.

They add that the students do not get direct feedback on their own solutions but are encouraged to discuss their work with other students based on the correct solution when it is presented; their rational is, “If students received the evaluations and corrections of their own solutions, they would study those evaluations instead of participating in the classroom discussion”. This drawing out of the students own reasoning applies part of the Socratic teaching approach discussed in the previous section, and
the “evaluate” phase between answering and the presentation of the correct solution makes efficient use of time.

They suggest that evaluation by peers may create discussion around readability – whether the variables have meaningful names and the code is indented correctly.

Informa can itself evaluate solutions “by comparing [them] to the correct solution provided by the author of a problem, or by automatically checking certain properties, such as whether a submitted piece of Java code compiles” (Hauswirth and Adamoli, 2013).

Hauswirth and Adamoli (2013) provide visual feedback in their system, explaining that the evaluation phase, filling the time between task completion and the revelation of the correct solution, asks students to provide feedback on their peers’ solutions, as well indicating how confident they are in their response. Figure 2.3 (Hauswirth and Adamoli, 2013), taken from their paper, shows a table, showing each student’s evaluation of each peer’s answer. This reflects only the boolean condition of correct or incorrect and shows student #1’s answer as incomplete.

According to the researchers, the resulting solutions can be analysed to determine from the proportion of correct answers and the distribution of completion times if the question is too easy or too hard, so that the problems presented can be refined for future lectures. Hauswirth and Adamoli (2013) state that, “a large IQR (a wide dispersion of the scores) is indicative of a useful problem”. This would reflect some students scoring highly and some lower, in accordance with their abilities. By taking a measure of the students’ confidence in their answers, the system provides further metrics which can be analysed. In addition, all edit operations are logged and timestamped, so that detailed analysis of the problem-solving process can take place.

Hauswirth and Adamoli (2013) conducted a case-study of their system, taking place over two semesters and using undergraduate students, consisting of lectures where the students read the material in advance and were encouraged to apply and discuss it in 20-40-minute Informa sessions, where appropriate to the material. They took data from 17 sessions, comprising 38 problems, with 641 solutions and 5029 evaluations. They found that 75% of peer feedback given in the evaluation phase is correct, and that there is a positive correlation between students’ confidence and the quality of their answer, but that students often don’t recognise their own weaknesses. They suggest the “Solve & Evaluate” approach as a potential remedy, with students hopefully discovering their own knowledge gaps when reading their peers’ solution.
They analysed the use of their system and suggest areas for improvement and further development. Of the suggestions proposed by Hauswirth and Adamoli (2013), the following are of note: a wider range of problem types; aggregation of solutions by common features, such as errors; better automated checking, improving on the current offering which only validates multiple choice answers, by validating solutions to which a simple answer, such as a type, is known; and allowing the testing of additional parameters or execution with a unit test; the development of a community where tutors can refine and share problems; and a web-based JavaScript version for increased portability.

2.2.3.2 Efficacy
Unfortunately, Hauswirth and Adamoli (2013) did not directly compare the learning outcomes of their system against those of no system, or those using other existing systems, so it is difficult to evaluate the efficacy of their approach.

They presented the students with a questionnaire; these students generally said they found the system useful. This data is subjective but does, however, provide justification for further investigation, measuring pre-and post-lecture knowledge between groups, as conducted by Thomas et al. (2015) into the use of a traditional SRS.

Hauswirth and Adamoli (2013) explain that the system can take advantage of anonymous answers by assigning each user a random and undisclosed integer, or it can afford psuedonymity, allowing the user to choose an arbitrary name which only they recognise. They add that these features are optional, and the lecturer can enforce the use of standard usernames. The researchers posit that, while being identifiable provides motivation to supply a good answer, anonymity reduces pressure on the students and ultimately results in better answers. A key benefit claimed is that Informa, “allow[s] the system to make use of humans for the evaluation” of solutions and they state that, at the time of writing and to the best of their knowledge, “no existing group response system has explored pedagogical scripts like ours, where evaluating the solutions submitted by peers turns the wait time of faster students into a valuable part of the learning process”.

2.2.3.3 Limitations
Hauswirth and Adamoli (2013) concede flaws in their approach, one of which is the need for students to configure their firewalls to work with the software. In a large class this could become extremely disruptive. With students using a variety of platforms, no single set of instructions can be formulated to resolve this issue, and it seems likely that some students with atypical set-ups will end up completely excluded from the task; this, however, is not explicitly stated in the report and remains conjecture. Informa requires the use of Java on both the server and client side.

They concede that the system can require significant set-up time at the start of a lecture, especially during the first few sessions; that all students require a suitable device connected to a network for the whole session; that the system has issues with scalability, with the number of evaluations rows limited
Chapter 2 Literature Review

and a lack of a means to aggregate similar/identical solutions; and that the timestamps are derived from the students’ individual devices and can therefore provide inaccurate data when showing the timings of the completion of tasks.

In the case study, they received feedback suggesting that weaker students may find the evaluation phase confusing. According to their report, in this phase students are required to grade other’s solutions from totally correct to totally incorrect (1-10), and if they are uncertain of the topic, and have no information on the validity of the solutions, it may be unhelpful to view several potentially wrong answers. They suggest carefully balancing the difficulty of the topic to counter this. Also discussed are the issues of students missing out on the evaluation phase if they finish last and the potential for peer evaluations to be inconsistent and/or inaccurate.

They concede further that the whole process to solve and evaluate a solution may take significant time, with some questions consuming more than 20 minutes of class time.

2.3.4 Other Systems

2.3.4.1 Overview

Kahoot! offers anonymous in-class participation, free and in a browser window, with the option of quizzes, multiple choice questions, and discussion questions (Plump and LaRosa, 2017). Plump and LaRosa (2017) make an interesting observation which seems applicable to SRSs in general, “The students seemed more comfortable asking questions when they could see other students got the wrong answer too.”

Plump and LaRosa (2017) state that the system cannot accept open-ended responses and questions have a character limit; both limitations severely limit the application’s usefulness in the domain of programming. Multiple choice questions are acceptable to augment free-form code generation questions, but do not serve as an adequate alternative.

Plump and LaRosa (2017) state that open ended questions may be coming soon. With this addition, the system would be comparable to Socrative in terms of functionality.

2.3.5 Summary

Whilst many of the systems described above have demonstrated efficacy in some domains, none of them can quickly evaluate, aggregate and provide feedback on programming problems.

Traditional SRS clickers suffer from a high unit cost of up to $60 per clicker (Taylor, 2013). They also require the installation of specialised infrastructure and are best suited to multiple choice questions.

Thomas et al. (2015) found no obvious benefit from using SRSs in their survey in which traditional systems were used; however, the courses were not designed with the SRS in mind and much of the
data were lost. This data loss meant that the lecturer could not as easily assess the class’s state of knowledge. Although they found increased engagement, knowledge attainment was unchanged. This shows that engagement is only valuable if the students are engaged in a useful task.

Modern SRS systems like Socrative provide a backchannel, by which students can anonymously ask questions, which allows students to communicate with tutors, asking questions they may otherwise be too nervous to ask (Guarascio et al., 2017).

Socrative provides a wide range of questions, including free-text. Programming solutions provided as free-text cannot, however, be automatically evaluated. Students using Socrative self-reported increased understanding of the material, and “enhanced the exchange of the information with the lecturer” (Dakka, 2015).

Free-form answers, such as code submitted by students are hard to evaluate automatically or in batch. The Informa system (Hauswirth and Adamoli, 2013), successfully implemented peer-evaluation in an SRS to provide feedback and fill in the down time between the first and last student answer. Students reported that they found the Informa system useful (Hauswirth and Adamoli, 2013).

Informa provides a good solution; however, some questions can consume up to 20 minutes, the feedback is not 100% accurate, and it uses a Java client (Hauswirth and Adamoli, 2013).

2.4 Recommendations for Further Activity

This literature review has explored several ways in which programming could be taught effectively, along with deficits in current practice. Current SRS systems, along with their efficacy, have also been explored.

Existing SRS systems provide an effective solution for education in many domains but are lacking when it comes to the automatic evaluation of free-form answers, required when creativity and problem solving are to be exercised.

Based on the points discussed, it would be useful to implement and evaluate a new SRS which offers problem types requiring the student to produce code to solve realistic problems and to generate unit tests. These solutions should provide immediate feedback to both student and lecturer.

The student should be afforded anonymity and the ability to amend their solution based on basic feedback, to direct them towards their own correct solution and to afford them freedom to fail in a safe environment.

The students should be ranked anonymously against their peers at the end of the session to provide an element of interest through competition and to help them rate their own relative level of understanding.
Chapter 3

Methodology

To test achieve the objectives and test the hypotheses, new features must be implemented in either a new system, or by extension of an existing one, and an evaluation session must take place.

Section 3.1 Requirements Analysis presents a set of requirements, elicited from the recommendations of the previous chapter, objectives, and some usage scenarios.

Section 3.2 Delivering the Requirements discusses potential means of implementation at a high-level – should an existing system be extended, or a new one created?

Section 3.3 Evaluation sets out a plan for the evaluation of the project and hypotheses.

3.1 Requirements Analysis

To test the hypotheses and achieve the objectives listed in Section 1.2.2 Objectives, I have developed some potential problem types to be implemented by the SRS system. Based around these, I have developed a set of requirements. These are broken down into functional requirements and non-functional requirements. Functional requirements specify what the system must be do, whilst non-functional requirements impose constraints on how the system must do it (Sommerville, 2016, p. 105).

The functional requirements are prioritised and split into mandatory and optional requirements.

In Table 3.1 in Section 3.1 Proposed Problems Types, I introduce two possible problem types. Due to the time constraints of the project, Problem Type 1 will be prioritised, and Type 2 is optional.

Section 3.1.2 Scenarios explores some usage scenarios. From these, the user requirements listed in sections 3.1.3 and 3.1.4 are derived. The implementation of these of these requirements across iterations is described in Section 5.6.3 Iterations.

3.1.1 Proposed Problem Types

Based on the ideas introduced in Chapter 2, I propose two problem types for a new SRS application:

1. Write a Class – Students are prompted to write a class fulfilling a given interface, which is then run in a sandbox against a suite of tests supplied by the lecturer. Feedback from the compiler and unit tests are then used to generate quantitative feedback. The lecturer provides the class outline, leaving blank spaces for the students to complete; in this way, the problem can be formulated as write a class, or, if the wider class structure is supplied, write a method
or part of a method. Excluded regular expressions may be added to enforce the use of particular structures.

2. Write Unit Test(s) – Students are prompted to write a unit test or set of tests looking for specific conditions. These tests are executed against an oracle and a set of mutants to provide feedback.

These problems can be set as individual or group tasks: as an individual task, students would use a device each and submit their own solution; as a group task, they would share a device and submit a group solution.

Problem Type 1 was prioritised over Problem Type 2. Problem Type 1 teaches core Java skills and, and as such is likely to be more heavily used than Problem Type 2. Problem Type 1 is also easier to implement, as it does not require populations of mutants to be provided and can be evaluated directly from the output of JUnit.

3.1.2 Scenarios

Lecturer’s Perspective 3.1.2.1
From a lecturer’s perspective the process begins when they sign-up and login. Next, they create a lecture, followed by an arbitrary number of instances of Scenarios 2 and 3 (if Scenario 3 is included in the deliverable), as they add problems. The lecturer then distributes the URL provided for joining the lecture and commences the lecture. The lecturer can terminate the lecture early, or it can end automatically when all students have submitted solutions for all problems. During the lecture, the lecturer can view the progress and results of their class.

Student’s Perspective 3.1.2.2
After being issued with the URL provided by the lecturer, the student joins the lecture. After the lecturer begins, the student is presented with the problems. They send solutions for evaluation, after which they receive feedback, and may submit the solution or attempt to improve it. After completing each problem, they are either presented with the next, or, if they have finished, wait until the end whilst viewing their personal results and the progress of their peers.

3.1.3 Functional Requirements
The functional requirements, grouped by priority and the user served, are listed below, with the objectives they fulfil listed afterwards. The priorities were produced in such as to map easily to iterations, delivering the core functionality quickly, reducing risk, and providing a structure to build further iterations upon.
For example, implementation of the priority 1 requirements provides enough functionality to perform evaluation of several of the hypotheses, and delivers the entities and infrastructure required to neatly implement the priority 2 requirements, in the next iteration.

General requirements for all users (GFR):

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Objectives Served</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFR1-End on Complete</td>
<td>After all students answer all questions, the lecture session must end.</td>
<td>Obj-Augmentation, Obj-Feedback, Obj-Progress</td>
<td>1</td>
</tr>
<tr>
<td>GFR2-Anonymise</td>
<td>Students must be anonymous.</td>
<td>Obj-Augmentation, Obj-Engagement</td>
<td>1</td>
</tr>
<tr>
<td>GFR3-Evaluate Immediately</td>
<td>Programming problem solutions must be evaluated immediately based on the amount of unit tests they pass.</td>
<td>Obj-Augmentation, Obj-Feedback, Obj-Progress</td>
<td>1</td>
</tr>
<tr>
<td>GFR4-Generate URL</td>
<td>Each course must supply a unique URL from which students can join a lecture.</td>
<td>Obj-Augmentation</td>
<td>2</td>
</tr>
<tr>
<td>GFR5-Evaluate Tests Immediately</td>
<td>The system could grade unit testing problems by awarding points for every error caught.</td>
<td>Obj-Augmentation, Obj-Feedback, Obj-Progress</td>
<td>5</td>
</tr>
</tbody>
</table>

*Table 3.1: General Functional Requirements*

Requirements for students (SFR):

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Objectives Served</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFR1-Display Problems</td>
<td>The program must display a series of Java programming problems to a cohort of students during a lecture.</td>
<td>Obj-Augmentation, Obj-Engagement, Obj-Java</td>
<td>1</td>
</tr>
<tr>
<td>SFR2-Display Next</td>
<td>If another problem exists, the system must present the student with next problem in the lecture after a solution is submitted.</td>
<td>Obj-Augmentation, Obj-Engagement</td>
<td>1</td>
</tr>
<tr>
<td>SFR3-Give Student Solution Feedback</td>
<td>After submitting each solution, the student must be assigned a grade (based on the number of tests passed) and presented with it, as well as a total for the whole lecture.</td>
<td>Obj-Augmentation, Obj-Feedback</td>
<td>1</td>
</tr>
<tr>
<td>SFR4-Allow Resubmit</td>
<td>The student must be able to resubmit multiple times. This affords the student freedom to fail.</td>
<td>Obj-Augmentation, Obj-Engagement</td>
<td>1</td>
</tr>
<tr>
<td>SFR5-Provide Student Summary</td>
<td>After submitting solutions for all problems, the student must receive a score based on their overall performance.</td>
<td>Obj-Augmentation, Obj-Feedback</td>
<td>2</td>
</tr>
<tr>
<td>SFR6-Rank Student</td>
<td>At the end of the lecture, students must be ranked anonymously against their peers.</td>
<td>Obj-Augmentation, Obj-Feedback</td>
<td>2</td>
</tr>
<tr>
<td>SFR7-Pose Anonymous Questions</td>
<td>The system must allow students to pose anonymous questions to the lecturer.</td>
<td>Obj-Augmentation, Obj-Interaction, Obj-Anonymity</td>
<td>4</td>
</tr>
<tr>
<td>SFR8-Retain for Revision</td>
<td>The system could retain anonymised session data for student revision after the lecture.</td>
<td>Obj-Augmentation, Obj-Feedback</td>
<td>5</td>
</tr>
</tbody>
</table>

*Table 3.2: Student Functional Requirements*
Chapter 3 *Methodology*

Requirements for lecturers (LFR):

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Objectives Fulfilled</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFR1-Commenge Lecture</td>
<td>The lecturer must be able to commence a lecture at a time of their choosing.</td>
<td>Obj-Augmentation, Obj-Interaction, Obj-Progress</td>
<td>1</td>
</tr>
<tr>
<td>LFR2-Create Account</td>
<td>The lecturer must be able to create an account and log in.</td>
<td>Obj-Augmentation, Obj-Progress</td>
<td>2</td>
</tr>
<tr>
<td>LFR3-Provide Lecturer Summary</td>
<td>At the end of a lecture, the lecturer must receive a breakdown of which students passed which tests as well as overall and average scores.</td>
<td>Obj-Augmentation, Obj-Progress</td>
<td>2</td>
</tr>
<tr>
<td>LFR4-Create Lectures</td>
<td>The lecturer must be able to create multiple lectures.</td>
<td>Obj-Augmentation, Obj-Progress</td>
<td>2</td>
</tr>
<tr>
<td>LFR5-Add Problems</td>
<td>The lecturer must be able to add multiple problems to a lecture.</td>
<td>Obj-Augmentation, Obj-Progress</td>
<td>2</td>
</tr>
<tr>
<td>LFR6-Specify Java Problem</td>
<td>The lecturer must be able to specify problems in the form of writing Java methods or classes, which students complete.</td>
<td>Obj-Augmentation, Obj-Progress, Obj-Java</td>
<td>2</td>
</tr>
<tr>
<td>LFR7-Provide Java Problem Test Suite</td>
<td>The lecturer must be able to provide unit tests for evaluating the solutions.</td>
<td>Obj-Augmentation, Obj-Progress</td>
<td>2</td>
</tr>
<tr>
<td>LFR8-Trigger End</td>
<td>The lecturer must be able to end the lecture session at an arbitrary time.</td>
<td>Obj-Augmentation, Obj-Feedback, Obj-Progress</td>
<td>2</td>
</tr>
<tr>
<td>LFR9-Specify End Time</td>
<td>The lecturer must have the option of specifying a time limit on a lecture.</td>
<td>Obj-Augmentation, Obj-Feedback, Obj-Progress</td>
<td>3</td>
</tr>
<tr>
<td>LFR10-Provide Skeleton Notation</td>
<td>Some notation, in the form of tags or labels, must exist by which the lecturer can specify where student input should be supplied in a partially written solution.</td>
<td>Obj-Augmentation, Obj-Progress</td>
<td>3</td>
</tr>
<tr>
<td>LFR11-Demonstrate Solutions</td>
<td>At the end of a lecture, the lecturer must be able to display solutions of their choosing for the class to see.</td>
<td>Obj-Augmentation, Obj-Interaction</td>
<td>4</td>
</tr>
<tr>
<td>LFR12-Add Regex Test to Java Problem</td>
<td>The lecturer must be able to add excluded regular expressions to a problem, to enforce the use specific structures.</td>
<td>Obj-Augmentation, Obj-Progress</td>
<td>4</td>
</tr>
<tr>
<td>LFR13-Specify JUnit Problem</td>
<td>The system could allow the lecturer to produce a JUnit testing problem, providing a suite of mutants whose errors must be caught by the solutions provided and an oracle to verify them.</td>
<td>Obj-Augmentation, Obj-Progress, Obj-Java</td>
<td>5</td>
</tr>
<tr>
<td>LFR14-Retain for Review</td>
<td>The system could retain anonymised session data for lecturer review, after the lecture.</td>
<td>Obj-Augmentation, Obj-Progress</td>
<td>5</td>
</tr>
<tr>
<td>LFR15-Aggregate</td>
<td>The system could aggregate similar solutions.</td>
<td>Obj-Augmentation, Obj-Progress</td>
<td>5</td>
</tr>
</tbody>
</table>

*Table 3.3: Lecturer Functional Requirements*
3.1.4 Non-Functional Requirements

The non-functional requirements, along with the objectives they fulfil, are as follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Objectives Fulfilled</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFR1-Be Reliable</td>
<td>The system must be reliable.</td>
<td>Obj-Augmentation Obj-Engagement Obj-Interaction Obj-Feedback Obj-Progress Obj-Anonymity</td>
<td>1</td>
</tr>
<tr>
<td>NFR2-Be Portable</td>
<td>The system must be highly portable – the client must be available using Firefox, Edge and Chrome on desktop; Safari on iOS; and Chrome on Android.</td>
<td>Obj-Portability</td>
<td>1</td>
</tr>
<tr>
<td>NFR3-Be Intuitive</td>
<td>The system must provide an intuitive interface – usable without prior training.</td>
<td>Obj-Augmentation Obj-Engagement Obj-Interaction Obj-Feedback Obj-Progress Obj-Anonymity</td>
<td>1</td>
</tr>
<tr>
<td>NFR4-Be Responsive</td>
<td>The system must be responsive – the system must provide immediate feedback on the status of submission evaluation and compile and evaluate solutions in a timely manner.</td>
<td>Obj-Augmentation Obj-Engagement Obj-Interaction Obj-Feedback Obj-Progress Obj-Anonymity</td>
<td>2</td>
</tr>
<tr>
<td>NFR5-Be Extensible</td>
<td>The system must be extensible, so new features and problem types can be added.</td>
<td>Obj-Augmentation Obj-Engagement Obj-Interaction Obj-Feedback Obj-Progress</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.4: Non-Functional Requirements

3.2 Delivering the Requirements

3.2.1 Extending Current SRS Systems

The Informa system, of Hauswirth and Adamoli (2013), provides an extensible SRS framework, tested in the domain of programming education, fulfilling NFR5-Be Extensible. It provides plugins for teaching various elements of the Java programming language; these involve getting the student to highlight syntactic structures; inferring types produced by an expression; drawing diagrams to reflect control flow structures; and a coding exercise, to bring together the first three disciplines, where the student completes skeleton code (Hauswirth and Adamoli, 2013). This fulfils LFR10-Provide Skeleton Notation and LFR6-Specify Java Problem.

Informa can evaluate solutions “by comparing [them] to the correct solution provided by the author of a problem, or by automatically checking certain properties, such as whether a submitted piece of Java code compiles” (Hauswirth and Adamoli, 2013). In this way Informa provides a rich set of question
formats and feedback, as well as manifold options for structuring a lecture around it, whilst fulfilling GFR5-Evaluate Tests Immediately.

The system’s client-side is developed in Java (Hauswirth and Adamoli, 2013); this creates problems with portability. Android and iOS have no Java Virtual Machine (JVM) implementation (Finger, 2013). For this reason, Informa fails to satisfy the requirement NFR2-Be Portable.

Whilst Hauswirth and Adamoli (2013) have made their system available under an open-source license, I have opted not to adapt their system, as the time required to understand the existing structure and reimplement the client-side using a more portable technology, would be prohibitive.

It would also require modification to support evaluation using JUnit tests and regular expressions.

3.2.2 A New System
By developing a new system from scratch, I am free to implement a solution using the technologies most suitable to the objectives and requirements of the project, free from the constraints of another system, and the requirement to become acquainted with its structure. This approach is cleaner, delivering a program whose codebase is entirely dedicated to the purposes of this project.

I have, therefore, elected to develop a new system. The design and implementation of this system are discussed in Chapter 4 Design and Implementation.

3.3 Evaluation
The evaluation of this system served three purposes: to evaluate the usability of the system, to evaluate the hypotheses, and to evaluate success of the project based on which objectives have been achieved.

3.3.1 Performance under Load – Responsiveness and Reliability
For the system to be adopted, and the objectives to be fulfilled, it is important that NFR4-Be Responsive is fulfilled. Key to this is the application’s responsiveness under load. This factor is also an important component of NFR1-Be Reliable, as the system must not crash or become prohibitively slow under heavy use.

To carry out this testing, a variant of the system was produced, in which JavaScript clients join a lecture presentation, and upon a trigger issued by the server, submit a solution for compilation and testing. Each session received is assigned a consecutive identification number, n, and submits its solution after a delay of n/10 seconds, using integer division; for example, 23/10 using integer division would yield 2. The result of this is that submissions are received in waves of 10 at intervals of 1 second.
Chapter 3 Methodology

As the client side receives its updates via AJAX, timing on this side is subject to a margin of error equal to that of the polling interval; therefore, the time taken from the receipt of a submission, to the preparation of the results for sending to the client, is recorded on the server side.

The results of this evaluation are recorded in Section 5.1.1 Performance under Load.

3.3.2 Usability
The usability of the system is another important factor in the systems adoptability, with NFR3-Be Intuitive mandating that the system be usable without prior training.

Brooke (1996) introduces the System Usability Scale (SUS), a Likert scale, to measure any system’s “appropriateness to purpose”, designed for low-cost usability evaluation. According to his article, this generalisation must be used, as the specific measures of the usability of a system are dependent upon its purpose. To measure this, he suggests using “subjective measures”. His SUS, fulfils this need.

Brooke (1996) describes his use of extreme examples and statements to provoke disagreement and avoid ambiguity, with a 50/50 split between questions which tend to elicit extreme agreement and extreme disagreement, to prevent “response bias”. Brooke (1996) recommends that his 10-point questionnaire be delivered before any discussion of the use of the system and to ask participants “to record their immediate response to each item”.

The SUS will be employed in the usability survey, as it delivers a short, simple, well-tested, off-the-shelf solution for usability evaluation.

3.3.3 Objectives and Hypotheses
Several questions were devised to evaluate the extent to which the objectives were achieved, and the truth of the hypotheses. These were placed in a section of the post evaluation session survey. These are discussed in the next section below.

3.3.4 User Evaluation Design
A user evaluation was carried out, student volunteers were asked to take part in an evaluation lecture and answer a series of questions based on their experience. As finding volunteers to participate at the same time, on site, in a classroom-based session was difficult, they were asked to take part individually and remotely, whilst imagining the system’s use as part of a lecture when answering the questions.

This required extra work, adding dummy students to the program to simulate other students taking part at the same time as the volunteers, as well as adding an extra option to keep the presentation open after all current attempts were finished.
Chapter 3 Methodology

3.3.4.1 Consent Form
The first page of the survey contains a consent form. The subjects were asked to complete this before taking part in the lecture session. Those who agreed to the terms, completed the lecture before moving onto the next session. The content of this form is shown in Section D.1 The Consent Form.

3.3.4.2 Student Details
After completing the lecture session, the subjects completed the Your Details form, as shown in Section D.2 The Student Details Form, followed conditionally by D.3 The Place of Study Form, asking those currently studying computer science about their place of study, and D.4. The Past Study Form asking those who are not currently studying the subject about their past study. The purpose of these forms was to determine the students’ gender and educational background, for the sake of analysing the data according to these groups, if enough volunteers participated.

All students were presented with a free-text box to give details about their programming experience.

3.3.4.3 Usability
As discussed in 3.3.2 Usability, the SUS of Brooke (1996) was employed to evaluate the usability of the system. Figure D.1 (Brooke, 1996) in Appendix D, shows the SUS Likert questionnaire. The questions listed were included in the 4th form presented to the subjects in the evaluation.

The section was introduced with the following text, asking the volunteers to imagine the system being used in a lecture setting:

“Please answer the following questions imagining the following: You are taking part in an introductory course and have been introduced to some elementary Java. During the lecture, your lecturer shares the link to the system and you connect to it and attempt to solve the problems presented alongside you classmates. In this scenario, to what degree do you agree with the following statements?”

3.3.4.4 Objectives and Hypotheses Likert
Likert scale questions related to the objectives and hypotheses, with responses 1-5 representing Strongly Disagree to Strongly Agree, were presented in the 5th survey section. Each question maps to an objective and/or hypothesis, and each question was paired with an inverted version of itself. This inversion was based on that described by Brooke (1996), to prevent “response biases”.

Questions which require a comparison with other systems were not marked as mandatory, as some students may not have experience of them. These were introduced in a similar manner to those of the usability study, asking the volunteers to imagine the system in a lecture setting, and are shown in Section D.6 Objectives and Hypotheses Likert.
Chapter 3 Methodology

3.3.4.5 Final Comments – Free Text Questions

The final form consists of a series of optional free-text questions, used to gain further insight into how successfully the objectives were implemented. These are shown in Section D.7 Final Comments – Free Text Questions and ask which features should be removed, retained, or added; and for a description of the platform used to execute the tasks as well as problems encountered. This information was used to evaluate the success to which Obj-Portability was achieved. The questions in this section were also used for formulating the recommendations in Section 7.1 Recommendations for Further Work.

3.3.4.6 Analysing the Results

3.3.4.6.1 Analysing the SUS

Brooke (1996) details how to derive a score from the SUS. His method is to take each items contribution of 0 to 4, \( c \), perform the operation \( c - 1 \) for odd numbered items, and \( 5 - c \) for even numbered items, and sum up the results, and multiply this by 2.5. He adds that the scores for the SUS range between 0 and 100.

An overall score of 68 is considered average, with scores above being above average, and those below, below average (Sauro, 2011).

3.3.4.6.2 Comparing two Groups with Mann Whitney U Test

To test the truth of a hypothesis, it is necessary to test the likelihood of the null hypothesis being true – the statistical significance. Glen (2015) states that the results of Likert surveys which contain categories whose distance apart cannot be determined, should be treated as ordinal values. As this survey will measure responses in categories such as “Agree” and “Strongly Agree”, this condition holds true.

Of the suggestions she lists for comparing the medians of populations of ordinal data, she describes the Mann Whitney U test, adding that, using this test, it is “[s]imple to evaluate single Likert scale questions”. This makes it a good solution for the evaluation of these hypotheses. For this evaluation, the populations shall be the responses received in the evaluation and a population representing the null-hypothesis for each question.

According to Minitab (2016), there is debate as to whether parametric or non-parametric tests are the best way to analyse Likert data. Performing a comparison between a parametric test, the t-test, and a non-parametric test, the Mann Whitney U test, to compare two groups, the author found little difference in the “statistical power”, with similar rates of false positives and negatives.

Similar to the example of Brooke (1996), the operation \( c - 1 \) was performed on the positive items and \( 5 - c \) on the negative items. The two groups were then combined into a single positive item, with double the original sample points.
Chapter 3 Methodology

The statistical significance of all of the items in Table D.5: Objectives and Hypotheses Likert was measured using the Mann Whitney U test, by comparing the results against a population representing the null hypothesis. The null hypothesis is represented by the value 2, as in the range 0 to 4, this represents the value Neither Agree nor Disagree.

As the respondents who provided genders categorised themselves as only male and female, producing two groups, I used this test to compare the results from Table D.5: Objectives and Hypotheses Likert across genders, as ideally the system would receive positive results across all genders, and if it not, the reasons would require investigation.

The results of these evaluations can be found in Chapter 5 Findings. An online Mann Whitney U calculator was used to perform these tests.

3.3.4.7 The Evaluation Sessions

Students were invited to volunteer to take part in a lecture session, after which, they were asked to complete the survey described above. As it was difficult to get students to commute to the campus and take part in a large lecture, the session took place remotely and students were asked to imagine the system’s use in a face-to-face lecture setting when evaluating the system.

The session involved the students joining a lecture presentation consisting of 3 problems, with 10 sets of randomised results added so the students could see how the cohort progress functioned.

The problems presented consisted of a simple calculator class, where students had to complete the bodies for sum, subtraction, and modulus methods; an absolute value class, where the students had to complete a method body to return the absolute value using an if statement; and the same problem again but requiring the use of the ternary operator.

The ternary operator problem for example, made use of unit tests and regular expression tests, organised into groups such that, 5 points would be awarded for returning the absolute value, and 5 points awarded for using the ternary operator to do it.

This variety of questions demonstrated the use of both types of testing: regular expression and JUnit and gave the volunteers a chance to see problems where partially correct answers are possible, alongside bars to compare their results to the simulated peers.

The problems posed were not particularly advanced. This was conscious decision, because I expected many of the volunteers to have little Java experience, and as this was taking place outside of a lecture, where the material would be taught, I didn’t want this to negatively skew the results.
Chapter 4

Professional, Legal, Ethical and Social Issues

4.1 Professional Issues

All relevant laws and guidelines will be followed in the implementation of this project.

Where code is taken from third-party sources and is not part of the official documentation and is not recognised as “common knowledge”, this will be cited in the code. Code which is considered “common knowledge”, such as the use of a class interface in one of the Java APIs, will not be cited.

This project may use 3rd party libraries. When used, their licenses will be fully complied with.

The code will be written to a high-standard, taking account of reusability, readability and security concerns.

4.2 Legal Issues

Any data collected from students, be it during evaluation sessions or from surveys will be anonymised. Students surveyed will be discouraged from entering personal, sensitive or confidential information into free-text boxes. Any data taken from lecturers will not be personal or sensitive in nature, so no significant issues arise.

Citations will be included for code used which is not “common knowledge”.

No unnecessary personal data will be retained by the system. Student data will be anonymised and data taken in the evaluation sessions, such as questions, interviews or discussions will be recorded anonymously, and no personal questions will be asked.

4.3 Ethical Issues

Citations will be included for code used which is not “common knowledge”.

4.4 Social Issues

No unnecessary personal data will be retained by the system. Student data will be anonymised and data taken in the evaluation sessions, such as questions, interviews or discussions will be recorded anonymously, and no personal questions will be asked.

Some students may find being ranked against their peers demoralising. This is mitigated by their anonymity. Whilst unfortunate, this kind of potentially negative feedback is necessary for students to be able to quantify their level of understanding.
Chapter 5 Design and Implementation

This chapter discusses how the system was designed and implemented.

Section 5.1 The Entity Classes introduces the data structures required to implement the system.

Section 5.2 The Interface describes the design of the user interface.

Section 5.3 Technologies introduces the technologies chosen and the justifications for these choices.

Section 5.4 Design Patterns describes some of the design patterns employed in the system.

Section 5.5 Software Development presents the software development methodology chosen, the system’s stakeholders, and an account of the development process, structured around iterations.

Section 5.6 Implementation ties together the previous sections, by presenting illustrative, but not comprehensive, examples of how the technologies and design patterns were implemented to deliver the required interface and functionality.

5.1 The Entity Classes

The key data classes, or entities, in the system are as in Figure 5.1. These are mapped to tables in the database. As the Java Persistence Architecture (JPA) takes care of this mapping, there is no explicit database design for this project.

The LecturerEntity represents a lecturer’s account and holds their login details. Attached to the LecturerEntity, are an arbitrary number of LecturePlanEntities, representing the lecture plans created by this lecturer.

The LecturePlanEntity, representing a lecture plan, functions as a blueprint for lecture presentations, containing a list of problems, encapsulated in ProblemEntities, to be presented in the lecture. To present a lecture, the lecturer creates a LecturePresentationEntity, which is then attached to the LecturePlanEntity.

Figure 5.1: The Required Entities.
Chapter 5 Design and Implementation

The ProblemEntity stores a list of tests as TestEntitys and test groups as TestGroupEntitys, as well as a unit tests file to evaluate solutions, a skeleton solution to display to the student and a model solution to compile the unit tests against.

Each test is assigned to a test group. A test group can only pass if all the tests contained within are passed. A test group may also have a dependency, a group which must also be passed for the test group pass. By setting the dependency to itself, a TestGroupEntity effectively has no dependency. TestEntitys are either mapped to a unit test or a regular expression, with subclasses existing for each case. When mapped to a regular expression, the test fails if the expression is found in the solution.

The LecturePresentationEntity keeps track of all attempts at a presentation as AttemptEntity objects and all anonymous questions posed as QuestionEntitys.

Each AttemptEntity contains a collection of SolutionEntitys containing the solutions submitted for each problem. It also keeps track of which problems have been solved in the current plan and whether all problems, as a set of ProblemEntitys, have been completed.

The SolutionEntity contains a ResultEntity keeping track of the passed and failed tests, from which the past and failed test groups are derived. It also stores the solution as character array.

The ResultEntity is also used to pass status messages around, from testing and compilation, as these are called asynchronously and polled when the client polls the state of an evaluation.

5.2 The Interface

This section describes the user interface. A full walkthrough of the lecturer’s interface can be found in Appendix A and Login and of the student’s interface in Appendix B. A brief summary of the interface is described below.

5.2.1 The Lecturer’s Interface

Lecturers must register and login with their credentials. Upon logging in, the lecturer is shown the Moderate Lectures page. From here they may manage their lecture plans, as well the presentations of these.

From the previous page they can reach the View Progress page for a given presentation, from which they can view individual student’s solutions, a summary of the class’s progress, and read any anonymous questions submitted.

From the Edit Lecture Plan page, the lecturer can modify a lectures name and description and edit its problem set.
Chapter 5 Design and Implementation

The Modify Problem page allows lecturers to supply new unit tests; student instructions; a skeleton solution, to show the student; and a model solution.

The Edit Tests and Groups page allows lecturers to create and edit tests; create and edit test groups, assigning their names, descriptions and scores; as well as to assign tests to test groups and to assign dependencies to test groups. The lecturer may add regular expression tests from this page. They may not, however, create unit tests here, as these are derived from the unit tests file.

5.2.2 The Student’s Interface

Students join presentations by navigating to the URL supplied by their lecturer. Upon joining, they are asked to supply their username, after which, if the presentation has commenced, they are presented with the first problem, else they are presented with a holding page until it has.

The student is then presented with the first problem, consisting of a problem title, instructions and a code window showing the skeleton solution, with only the specified lines mutable and highlighted.

An Evaluate Solution button is provided to send solutions for evaluation. Upon receiving the result, the page scrolls down to display it. These results contain a colour-coded row for each test group, indicating pass or fail. Alongside these are progress bars, displaying a summary of the results of the student’s cohort, with the statuses of pass, fail and not attempted.

If the student has failed a test group, feedback is displayed beneath the corresponding test group row. An option is provided to allow the student to try again, which scrolls to the solution input area. From this section, they may also ask an anonymous question, which will be sent to the lecturer.

Upon on completion, the student will be shown their placement and a summary of their progress in the lecture, along with that of their cohort. Their placement is shown in a histogram displaying the distribution of scores across a number of ranges, with the bar representing their range marked. Their performance on each test group of each problem is shown below their placement, with bars representing their cohort showing the class’s totalled results on the same group. This is updated live to reflect their cohort’s progress.

5.3 Technologies

The application requires the display of lecture presentations on students’ devices, taken from a common source, as well as the receipt and evaluation of students’ solutions, followed by their distribution back to students’ and lecturers’ devices. For this reason, I have opted to use a client-server architecture for the system; the discussion is divided along these lines.
5.3.1 The Client-Side

5.3.1.1 Portability

This program requires a front-end interface to present and elicit information from the user and must be available on a wide range of devices.

HTML5 and JavaScript are a good candidate for this: “All modern browsers” provide support for HTML5 (w3schools, n.d.-b); the same is true of ECMAScript 5, the 2009 edition of JavaScript (w3schools, n.d.-a); this is essential to satisfy NFR2-Be Portable.

JavaScript is a high-level scripting language, executed by the user’s browser, which allows common actions, such as creating and retrieving data from dialogue boxes, to be carried out in single lines of code (Tanenbaum and Wetherall, 2010). Tanenbaum and Wetherall (2010) also describe the Document Object Model (DOM) as a tree by which a JavaScript script can access any element in a page and manipulate it.

Shahzad (2017) published a paper describing how HTML5, along with CSS, JavaScript and AJAX can be used to develop “Modern and Responsive Mobile-enabled Web Applications”.

He asserts that HTML5 can produce “fluid and responsive”, native-like applications, running on a single page. According to his paper, many JavaScript libraries exist, implementing commonly required functionality, providing frameworks and offering “nice-looking” UI components.

Due to the above considerations, HTML5 and JavaScript were used to implement the client-side interface.

Another option was Java, which allows the programmer to write programs, targeting the Java Virtual Machine (JVM), which is then responsible for executing these programs on the host (Sridhar, 2017).

Java would seem to be a good candidate for the client-side implementation; however, according to Hughes (2017), Google announced the dropping of support for Java Applets from its December 2017 Chrome release. As of December 2017, Chrome represents 77% of all browsers used (w3schools, 2017), so this is of importance. Finger (2013) states that the JVM is neither supported by iOS nor Android. Given that iOS and Android accounted for 17.9% and 81.7% of smartphones sold in Q4 2016 respectively (Gartner, 2017), this severely undermines the platform’s portability, and as such, was rejected as a viable option.

5.3.1.2 Asynchronous Updates

The client’s interface needs to be updated in an asynchronous manner when new data is available. This takes place in the following situations in the student interface: during the compilation and testing of solutions, the current status is displayed; whilst attempting the lecture, students are presented with their cohort’s progress on the current problem (Section B.2 Taking Part); and when a student has
completed a lecture, they are presented with a breakdown of their cohorts results, dynamically updated whilst other students are still submitting solutions (B.4 Completion – Seeing Your Results). This is also used in lecturer’s interface to display the current state of each presentation and to view live progress of their class (Section A.4 Moderating Lectures).

I considered two major technologies available for this task: AJAX and WebSocket. Although WebSocket is more efficient (Koning, 2014), AJAX appears to be a simpler technology to understand and implement; therefore, I opted to use AJAX to reduce potential risk and to cut down implementation time.

According to Tanenbaum and Wetherall (2010), Asynchronous JavaScript and XML (AJAX), is a collection of tools which can be used to allow pages on the client’s machine to communicate with a server via Extensible Markup Language (XML) messages. They list these technologies as HTML and CSS for presentation; DOM for manipulating elements in the page; XML as a language to communicate with the server; and JavaScript “as a language to bind all this information together”. According to Tanenbaum and Wetherall (2010), key to the usability of AJAX is that it allows “asynchronous I/O”, allowing the interface to remain responsive while it awaits data from the server.

According to Rocheleau (2016), AJAX allows a page to push data to the server, and, using JavaScript, parse received data without refreshing the page. Koning (2014) describes a potential bottleneck arising from the need to poll the server constantly for updates, producing overhead in setting up new connections.

Using HTML5 to define the page content as well as AJAX to update the pages, the page can be updated and can update data in a persistent database, without reloading the page (Shahzad, 2017).

AJAX is supported by the desktop variants of Edge, Firefox, Chrome and Safari, as well as Chrome for Android and Safari for iOS (Can I use, 2018).

JavaScript, as part of the suite of technologies making up AJAX, offers a viable platform upon which to build this project, with asynchronous communication between the client and server, allowing a responsive and up-to-date application to delivered.

5.3.2 The Server Side

5.3.2.1 Dynamic Web Pages

The program needs to dynamically generate web pages, showing different content for different users, as well as being to send updates, and received new data, such as solution submissions.

The points discussed above do not preclude the use of Java on the server-side, as long as no Java is required on the client-side, and Java is used to interface with non-Java technologies.
Rouse (2017) describes the Java Platform, Enterprise Edition (Java EE) as “a collection of Java APIs owned by Oracle that software developers can use to write server-side applications”. Rouse (2017) explains that the platform offers JavaServer Pages (JSP) which can handle common features such as input forms and cookies. She expands that the platform can handle persistence through the Java Persistence API (JPA), business logic in Enterprise Java Beans (EJB) as well as providing all the APIs included in Java Platform, Standard Edition.

Scholtz (2010) describes the relationship between JSP and Servlets. He describes JSPs as web page templates containing HTML and JavaScript, with dynamic content inserted within. He adds that JSPs are ultimately compiled into Servlets; in this way they are an abstraction of Servlets.

Oracle (2013) define a Servlet more specifically as a means of adding functionality to a server in a request-response model, with an HttpServlet being a specialised HTTP implementation. They add that this HttpServlet provides methods for handling HTTP requests.

Java was chosen for these reasons to implement the server-side program, but without the use of EJBs owing to a lack of support in the standard Tomcat server. Another concern was that of productivity – I have more experience with Java than other platforms, and as such, could develop the system more rapidly.

5.3.2.2 Persistence
The system needs to be able to store lecture plans and deliver them, as well as keep track lectures and attempts at presentations of lectures. For this reason, persistence in the form of a database is required.

According to Vogel and Scholz (2017), the Java Persistence Architecture (JPA) performs object-relational mapping, automatically handling the persistence of Java objects and the creation of tables, so the developer does not need to work at the level of SQL statements.

Crume et al. (2006) introduce Entity Beans as a means of abstracting the implementation of persistence in enterprise applications, with transactions and database connections being handled by the server. These Entity Beans are more commonly referred to as Entity Classes.

Vogel and Scholz (2017) add that classes can be made entity classes using an annotation; each of these is then mapped to its own table. They add that each of these entities is mapped to a table in a relational database, with the option of the JPA automatically generating primary keys.

They describe a means by which relationships between entities can be mapped, containment, where entities retain references to other entities, on a bidirectional or unidirectional basis. They describe unidirectional relationships as relationships where only one entity stores a reference to the other, and bidirectional as relationships where both entities mutually store references to each other.
According to their article, the JPA provides annotations defining one-to-one, one-to-many, many-to-one and many-to-many relationships.

The JPA was chosen for the implementation of persistence for the reasons described above.

5.3.2.3 Security
The two most obvious vectors of attack are query injection, and the submission of malicious code in a lecture presentation. Measures needed to be taken to mitigate these risks.

The program performs queries using the Java Persistence Query Language, which performs a similar roll to that of SQL. User data, such as the lecturer’s email address on the Lecturer Login page (Section A.1 Register and Login), are used as components of these queries. Where user data forms parts of a query, parameters are defined in the String, and the arguments bound at runtime. This is more secure than the more obvious option of using String concatenation, which opens up the possibility of an injection attacks.

The risks posed by the execution of arbitrary user code were mitigated by executing the tests in a separate process, preventing direct access to the Tomcat server instance, whilst invoking the new Java process with a policy file, limiting the user code’s permissions to access of its own declared members, and forbidding dangerous operations such as file system access.

5.4 Design Patterns

5.4.1 Extensibility
The program must be extensible to allow developers to add functionality. To enable this the program structure must be easy to understand, with clear demarcations between areas of functionality. The use of design patterns affords this, and by promoting common designs across programs, aids developers new to the project by exhibiting structures with which they already familiar.

Ladas (2007) introduces the concepts of coupling and cohesion. He describes coupling as the extent to which parts of a program are dependent upon others and cohesion as the extent to which the functionality is consistently divided up. In his words, “Cohesion measures the semantic strength of relationships between components within a functional unit”. This would imply that to satisfy the criteria of high cohesion, a class representing a hotel room would not contain information such as the occupants’ phone numbers, constraining itself exclusively to the attributes which describe a hotel room and the methods required to operate on them.

He adds that cohesion and coupling are often related, with high cohesion yielding low coupling, an outcome which is desirable.

This application will comprise a user interface, program logic and a database to provide persistence. Leff and Rayfield (2001) discuss one design pattern useful for organising these tiers: the Model-
Chapter 5 Design and Implementation

View-Controller (MVC) pattern. According to their paper, this pattern “decouples” the user interface from the application’s data by separating the UI, or “Presentation” from the data, or “Abstraction”, with the controller acting as a layer to facilitate communication between the two.

According to them, the advantage of this architecture is that the user interface can be modified without affecting the data representation or business logic. This is a benefit of loose coupling.

Specifically discussing the implementation of a web application using Java on the server side, they suggest using Enterprise JavaBeans as the model, JSPs for the view and “Servlets and Session EJBs” for the controller.

Separating the model, views and controllers also produces higher cohesion, with these discrete units of functionality being separated into different modules.

The use of the MVC pattern improves maintainability and extensibility, satisfying NFR5-Be Extensible.

According to Vogel (2016), the observer pattern consists of one subject to many observers, with the observers watching the subject for changes in state. He expands, that an unlimited number of observers can register with a subject, without necessitating change in code, decoupling the subject from the observers.

His examples show a subject storing a list of observers, with the observers implementing a common interface, allowing any class which implements this interface to register as an observer. This has clear advantages, in terms of maintainability and extensibility, helping to fulfil NFR5-Be Extensible.

5.5 Software Development

5.5.1 A Flexible Development Methodology

To develop the project, I needed a methodology that would allow me to revise the design as I experimented with the technologies, and discovered new requirements, and shortcomings in my initial approaches. This was due to my lack of experience with some of the technologies, and of designing large systems and educational tools.

Agile development methodologies afford this flexibility, and for this reason I adopted an agile approach to the development.

5.5.2 Stakeholders

Lecturers and students are the key stakeholders of this project, and as such, my project coordinator became the de-facto customer of the project. During our meetings I received feedback, from a lecturer’s perspective, on the system as it was under development.
Before development, requirements were elicited based on the objectives and the need to test to the various hypotheses and assigned priorities which mapped broadly to iterations. Each iteration involved informal planning and design, based on the requirements to be implemented; development, implementing the requirements in code; and testing of the new features, alongside testing to ensure old functionality still worked as expected.

5.5.3 Iterations

As I was unfamiliar with many of the technologies to be used, the requirements were prioritised and allocated to iterations according to two criteria: ensuring the most important functionality was delivered first; and allowing me to build up the core structure of the application gradually, whilst experimenting with and learning about the new technologies. These iterations are described below.

5.5.3.1 The First Iteration – Communication

In this iteration, I implemented a basic program which took input from the user, posted it asynchronously using AJAX to a Servlet, which persisted it using the JPA. I also implemented a simple counter which queried the Servlet via AJAX, showing the increments, which I eventually removed, keeping its core for my polling functions. In this way I had working examples of communication between all tiers, using all the required technologies, to adapt for their real purpose in the later iterations. I experimented with Enterprise Java Beans (EJB), before dropping them, due to a lack of obvious benefit and the complexity required in deploying them to Tomcat servers.

5.5.3.2 The Second Iteration – Present a Lecture

In the second iteration I produced a series of test problems and a test lecture, hardcoded into a Servlet for easy deployment. During this iteration, I decided upon the structure of the problems and the nouns and verbs to be used throughout the system. Using this lecture, I developed a JSP page, which presents the student with a series of problems and displays their final grade at the end based on evaluations using JUnit tests. Finally, I added the ability to start and stop lectures.
5.5.3.3 The Third Iteration – Lecturer Interface

In the third iteration I added functionality to allow lecturers to register and create their own lectures; add problems to these lectures; and to provide skeletons, specifying mutable areas, and model solutions for these. I also added the ability to view the progress of the class, during and after the lecture.

5.5.3.4 The Fourth Iteration – Loose Ends

The fourth iteration was open ended and dealt with cleaning up any missing mandatory requirements; fixing errors; code refactoring, to improve the readability, reliability and extensibility of the code; implementing any new ideas which emerged during previous iterations; and making various improvements suggested by my project coordinator.

Shortcomings in my implementation of persistence using the JPA emerged, with references to non-persisted objects existing in collections, giving the illusion of persistence, and the failure of the system to remove orphaned entities. I resolved this by learning in more detail how entities are managed and ensuring that both ends of bidirectional relationships were kept consistent.

I added improvements to the lecturer interface. Key among these was better input checking and preventing the lecturer from changing a plan once it had been presented. Although not exhaustive, the input checking makes the system more robust. To allow the alteration of presented lectures, I added functionality to the system to duplicate lectures, allowing the amendment of the new copy.

The ability for the student to pose anonymous questions to the lecturer was an added in this iteration. These appear in the lecturer’s progress view, discussed in the next section, alongside the problem to which they relate.

After comments from my project coordinator, I changed the rankings at the end of the lectures to more neutral placements; made the interface clearer; added indicators of the class’s progress on each problem and a breakdown of the class’s results in each test group; and used more neutral colouring and language to describe the student’s performance, removing red for fail and green for pass.

My project coordinator had commented on the use of regular expressions to evaluate code, so I added these to the fourth iteration (LFR12-Add Regex Test to Java Problem). To augment this functionality, I added the ability to set a dependency on a test group – allowing the lecturer to set test groups to pass only if another group also passes.

It became apparent that it would be useful to allow a lecture to stay open after the final attempt is complete, so students could carry out an attempt at a later date. This was especially useful for the evaluation, so emails could be sent out to allow students to join at any time. This was added to the requirements list.
The final task undertaken was to refactor the code to more faithfully implement the MVC pattern, and to increase cohesiveness and clarity.

5.5.3.5 The Fifth Iteration – Future Work

The fifth iteration is planned for future work, discussed in Chapter 9.

5.6 Implementation

5.6.1.1 Joining a Lecture

The system implements the MVC design pattern, albeit without using EJBs as discussed in Section 5.4.1. The view consists of HTML pages, enriched with JavaScript and generated by JSPs, the controller of Servlets to prepare data for and receive data from the view, and the model of Entity Classes to represent data and handle update and retrieval operations.

Figure 5.3 shows an abridged account of the communication between the tiers of application when a student navigates to the presentation URL, as a sequence diagram; the full diagram is available in Figure C.1 in Appendix C. The diagram shows communication between AttemptProblem.jsp, the view, operating on the client-side; the controller, consisting of the AttemptLecturePresentation and PollLecturePresentation Servlets; and the model, consisting of AttemptEntity and LecturePresentationEntity. PersistenceUtilities is used to simplify access to an EntityManager, required for persistence, and the HttpSession object is used as a key-value store to associate objects with the session.

The presentation’s join link points to the AttemptLecturePresentation Servlet, mapped to the path /join for brevity, which creates a new attempt if one doesn’t already exist for the requested presentation. The presentation to join is defined by the id parameter in the URL.

Not shown in the shortened diagram, is after the client is forwarded to the JSP, the pages JavaScript code sends an AJAX request, using jQuery to check if the student’s name is set, and if not, displays a form asking for this information.
The client’s JavaScript now begins polling the PollLecturePresentation Servlet to see if the presentation has begun, whilst displaying a holding message. Once it is has, it receives the first problem’s data and displays it. If an attempt is already in progress for this session and this presentation, the current problem is displayed.

The structure of the JavaScript employed is shown below. TechOctave (2011) addresses the risk of AJAX queries queuing up due to lag and arriving in the wrong order, by using recursive polling, with the polling function calling itself from within the POST callback. They add, however, that this comes at the expense of a guaranteed polling interval. This method is employed in the code.

```javascript
01  function pollLectureStart() {
02      var posting = $.post("PollLecturePresentation", {
03          action: "poll_start"
04      });
05
06      posting.done(function(data) {
07          if($(data).filter("#lecture_state").html() === "ACTIVE" ||
08              $(data).filter("#lecture_state").html() === "TERMINATING") {
09              ...
10          } else if($(data).filter("#lecture_state").html() === "TERMINATED"){
11              ...
12          } else if($(data).filter("#lecture_state").html() === "ATTEMPT_COMPLETE"){
13              ...
14          } else {
15              setTimeout(function() {pollLectureStart();}, 2000);
16          });
17      }
```

On line 2, jQuery is used to asynchronously send a POST request to the Servlet. The action parameter of poll_start indicates to the Servlet that the client is enquiring as to whether the presentation has commenced.

If the presentation is already terminated, this information is communicated to the client and the JavaScript program displays a message to this effect. If the user has already completed the presentation, they are shown their final scores.

On line 6 a callback is defined and passed to the posting object. This is executed when the POST request returns. The chain of if statements handles the possible presentation states. If none of the explicitly tested states is true, the state must be WAITING. This means that the presentation is still to be commenced, so a timeout is set to poll again.

5.6.1.2 Displaying Problems and Evaluating Submissions

After the first problem has been received by the polling loop described in the previous section, it is displayed to the student.
This screen consists of a problem title, *Ask a Question* button, instructions, solution input box, *Evaluate Solution* button, textual state information, results and cohort progress area, and finally the options to *Submit Solution* or *Try Again*.

Figure 4.4 shows the instructions for a problem, and a partially successful attempt. When the student clicks *Evaluate Solution*, their solution is sent to the EvaluateSolution Servlet for evaluation. The client-side JavaScript then polls this Servlet for updates, presenting them to the student. This polling follows the same pattern as that described above.

An abridged account of this communication is shown in Figure 4.5, with the full diagram included in Section C.2 *Sending a Solution for Evaluation*. The TestUtilities call returns immediately, after starting a separate thread, the progress of which is stored in the ResultEntity.

![Figure 5.4: Instructions and Solution Input](image)

![Figure 5.5: Sending a Solution for Evaluation (abridged)](image)
This new thread gets the system compiler statically from the ToolProvider class. The result of this is stored in the ResultEntity, along with any diagnostic information returned by the compiler. If the compilation fails, all tests are failed and the offending lines are highlighted.

The results of evaluating the above solution are as in Figure 5.6. Since the solution successfully returned the absolute value, the test group Return absolute passed; however, as this was delivered using the if statement, the Use ternary group fails. Use ternary is a test group consisting of regular expression tests excluding such conditional statements.

As Use ternary is dependent on Return absolute, one could not simply enter return 0 to pass it. Whilst this would pass the regular expression tests, the failure of Return absolute precludes the group’s passing. After evaluation, the student is free to amend their solution or submit as is.

Also included in the results area, are the colour coded cohort progress bars. These are generated in the PresentationEntity, which receives notifications of updates from the AttemptEntitys, as per the Observer pattern. A timestamp stored in the session is compared with a timestamp stored in the PresentationEntity to ensure this data is only sent when updates exist.

These cohort progress bars are derived from a map, mapping TestGroupEntitys to SummaryEntitys. A SummaryEntity encapsulates progress statistics of the cohort in a particular test group – passes, fails and non-attempts.

5.6.1.3 Placement

When the student completes their attempt, they are presented with their placement, as in Figure 5.7. This placement is subject to change if the presentation is still ongoing, so the JavaScript program continues polling for new updates in a fashion similar to that described in the previous section. This polling ceases once the presentation completes.
Progress bars for each test group indicating the student’s pass or fail status and the cohorts progress are displayed beneath the output in Figure 5.7 (not shown), in the same style as those shown in Figure 5.6.

5.6.1.4 Observing a Lecture

The example above was submitted as part of a presentation of a single-problem lecture plan, with 10 dummy attempts. The final cohort progress view, as displayed to the lecturer is shown in 5.8. The data is refreshed using AJAX polling; however, the functionality of this page is in some ways less sophisticated than that of the live progress bars described above: all data is sent as a response to every polling message and polling continues after the termination of the presentation. This is inefficient, but as there are fewer lecturers connected than students, this has less impact. These inefficiencies could be resolved by only sending data when updates exist, using timestamps to find out when the session last received an update and by testing the presentations status using JavaScript, as is used in the student’s progress view described above.

5.6.1.5 JUnit Evaluation

The paths used in the application are handled by the PathUtilities class. CompilationUtilities is passed a path derived from the IDs of the attempt, the problem, and the presentation.

```java
CompilationUtilities.compile(rawSolution, "Solution", PathUtilities.getInstance().getCompilationRoot() + "/presentations/" + attempt.getLecturePresentation().getId() + "/prob/" + attempt.getCurrentProblem().getIdInPlan() + "/solutions/attempt/" + attempt.getIdAsHex(), result);
```

Inside CompilationUtilities the classpath is produced, using the provided paths and the paths stored in PathUtilities.
Chapter 5 Design and Implementation

PathUtilities paths = PathUtilities.getInstance();
classPath += PATH_SEP + paths.getJUnitPath()
    + PATH_SEP + paths.getHamcrestPath()
    + PATH_SEP + compilationPath + SEP;

The classpath is added to a List of Strings called options and passed to the compiler, which then compiles the solution.

JavaCompiler.CompilationTask task = compiler.getTask(null, null,
    diagnosticCollector, options, null, compilationUnits);
boolean success = task.call();
result.setCompilationDiagnostics(diagnosticCollector.getDiagnostics());

Having a common store for compilation roots and dependency paths (such as the JUnit jar file), means that paths can be added to class paths from anywhere in the program, and compiled classes can be written to known locations.

In the above code, compilation diagnostics are stored in the result, so that if compilation fails due to a compile time error, the line number can be extracted and the student can be warned.

If the solution compiles successfully, this information is stored in the result.
result.setState(ResultEntity.State.COMPiled_AWAITing_Tester);

The solutions are then executed. The classpath is the created, so that Java knows where to find all of the required files. These include JUnit, Hamcrest, the solution class and the tests class.

PathUtilities paths = PathUtilities.getInstance();

String classPath = paths.getJUnitPath() + PATH_SEP +
    ...   "org.junit.runner.JUnitCore", "SolutionTest"};

Process proc = Runtime.getRuntime().exec(commands, null, javaPath);

The code above executes the tests as a separate process, so that the student’s code does not have access to the Java EE server, and passes a security policy file, limiting the code’s privileges exclusively to accessing its own declared members, preventing dangerous activity like file system access.

No user provided data is passed into the command String. This prevents injection attacks in this code.

The output of running JUnit is parsed, using a regular expression to catch failures. The corresponding TestEntitys are then marked as failed.
Chapter 6 Results

```java
if(resultString.contains("FAILURES!!!")) {
    result.setHasFailures(true);

    Pattern failurePattern = Pattern.compile
        ("[0-9]+\ \[a-zA-Z0-9]+\ \(SolutionTest\)\);  
    Matcher matcher = failurePattern.matcher(resultString);

    String current;
    while(matcher.find()) {
        current = matcher.group();
        current = current.substring(current.indexOf(""))
        + 2, current.indexOf(""));

        result.addFailure(attempt.getCurrentProblem()
                .getTestByName(current));
    }
```

Chapter 6

Results

6.1 Findings

6.1.1 Performance under Load

6.1.1.1 Results

The tests described in Section 3.3.1 Performance under Load were run on a third-party server – a virtual private server, online, with 4GB ram and 4 2.4GHz cores, running Tomcat 8.5.30 on Windows 2012. 1 wave of 9 submissions and 6 waves of 10 submissions were used to submit solutions for evaluation; this odd numbering is the result of a minor error which meant the system began counting at 1 instead of 0, leaving only 9 attempts in the first wave – I did not feel, however, that this significantly affected the outcome, so did not repeat the tests. Figure 5.1 shows these waves.

![Figure 6.1: Submission Waves](image)

Solutions were evaluated for a problem consisting of 3 JUnit and 4 regular expression tests, so were of moderate complexity. The average time from the receipt of a solution to the dispatch of results was 17641.84ms. The results had a standard deviation (σ) of 11282.77, indicating a large spread of results; indeed, the shortest evaluation took place in the 6th wave, taking 718ms, and the longest in the 1st wave, taking 38444ms.
Table 6.1: Average Evaluation Time by Wave

<table>
<thead>
<tr>
<th>Wave</th>
<th>Average Time Taken (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5522.89</td>
</tr>
<tr>
<td>1</td>
<td>14615.80</td>
</tr>
<tr>
<td>2</td>
<td>19874.50</td>
</tr>
<tr>
<td>3</td>
<td>23796.00</td>
</tr>
<tr>
<td>4</td>
<td>26203.80</td>
</tr>
<tr>
<td>5</td>
<td>20428.20</td>
</tr>
<tr>
<td>6</td>
<td>11839.80</td>
</tr>
</tbody>
</table>

Figure 6.2 and Table 6.1 show the average evaluation time by wave. It is interesting to note that the evaluation times peak around wave 4. The probable cause of this that the backlog peaks around the middle, as evaluations from wave the previous evaluations are still waiting and further evaluations from the next waves are still arriving. All threads wait on the same synchronised methods, and access is granted randomly, so submissions from wave 4 must compete for resources not only with those of the previous waves, but also those of waves 5 and 6. The raw results can be found in Section E.1.

6.1.1.2 Analysis

Whilst the wait times peak of an average of ~26 seconds for the 4th wave represent an unacceptable delay; the testing conditions were extreme compared to what could be expected in a real-world scenario. Real students take varying amounts of time to complete the same task, and it is unlikely this repeated concurrent submission would actually occur the real world. During these extreme conditions, the system remained stable and the average evaluation time was at an acceptable ~11 seconds. For these reasons, I would assert that NFR4-Be Responsive was implemented successfully.

The more extreme delays could be rectified by implementing a queue, so that solutions are evaluated in order of arrival, keeping the average time the same, but giving them fairer treatment.

These load tests could be improved by using a dedicated server, using more realistic usage patterns, and performing them several times to take averages.

6.1.2 User Evaluation

7 volunteers took part in the user evaluation. Unfortunately, this is a low number, so these results are far from conclusive. 2 students identified as female, 4 as male, and 1 did not provide a response.

All of the students are currently studying or have studied at Heriot-Watt and all were either studying or had studied a computer science related subject, so results from volunteers with computer science and none computer science backgrounds cannot be evaluated.
Chapter 6 Results

6.1.2.1 Objectives and Hypotheses

Questions which required the user to compare the system directly with other SRSs were marked as optional, to account for the possibility that some students may not have used other systems. Because the responses to the negative questions are aggregated with their positive counterpart, the counts shown are in fact double that of the original responses.

6.1.2.1.1 Engagement

Sys-Engage- and Sys-Engage+ together evaluate Hyp-Engagement and Obj-Engagement. Table 6.2 shows the results of these items, after inverting the negative item and combining it with the positive. 0 represents Strongly Disagree and 4 represents Strongly Agree. 6 volunteers provided responses for these items, which after adding the inverted responses, produced 12 responses to a single item.

6.1.2.1.2 Understanding

Sys-Understand- and Sys-Understand+ together evaluate Hyp-Understanding. Table 6.3 shows the combined results for these items. 6 volunteers provided responses for these items, producing 12 responses when combined into the positive question.
Chapter 6 Results

The average result, 1.75, would appear to suggest an overall negative response; however, the Mann Whitney U Test, yielded a statistically-insignificant p-value of 0.74896. It is not possible from this data to reject the null hypothesis, and, as such, the truth of Hyp-Understanding cannot be evaluated.

6.1.2.1.3 Feedback

Sys-Feedback- and Sys-Feedback+ together evaluate the truth of Hyp-Feedback Student and the status of Obj-Feedback. Table 6.4 shows the combined results for these items. 7 volunteers provided responses for these items, producing 14 responses when combined into the positive question.

Looking at the average, 2.28, the results appear to be positive; however, conducting the Mann Whitney U Test, yielded a statistically-insignificant p-value of 0.11184. It is, therefore, not possible to reject the null-hypothesis, or assess the truth of Hyp-Feedback Student or whether Obj-Feedback has been fulfilled.

6.1.2.1.4 Interaction

Obj-Interaction is evaluated by Sys-Interact Lecturer-, Sys-Interact Lecturer+, Sys-Interact Peers- and Sys-Interact Peers+. 7 people replied to each item, yielding two combined items of 14 responses.

The combined results for Sys-Interact Lecturer-, Sys-Interact Lecturer+ can be found in Table 6.5.

The average result, 2.5, for the Sys-Interact Lecturer combined item suggests a positive result; however, the Mann Whitney U Test yielded a statistically insignificant p-value of 0.20766.

The combined results for Sys-Interact Peers and Sys-Interact Peers+ can be found in Table 6.6.
Chapter 6 Results

Table 6.6: Sys-Interact Lecturer Combined Results

<table>
<thead>
<tr>
<th>Scale</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sys-Interact Peers/ Sys-Peers Lecturer+</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>14</td>
</tr>
</tbody>
</table>

The average result for the Sys-Interact Peers combined item, 2.21, hints at a slightly positive result; however, the Mann Whitney U Test yielded a statistically insignificant p-value of 0.11184.

As the results were insignificant, it is not possible to evaluate the fulfilment of Obj-Interaction.

6.1.2.1.5 Perceptions by Gender
Performing the Man Whitney U Test to compare the two provided genders, combined in the same fashion as above, none of the items returned a statistically significant p-value, so there is no significant difference in perceptions of the system between genders.

6.1.2.3 Usability Survey
The complete results of the usability survey can be found in Section E.2 Usability Survey Results. Averaging the scores from all responses yielded 76.79, above the score of 68 suggested by Sauro (2011) for an average system. This indicates that the system exhibits good usability, with the caveat that the sample size was very small.

The average SUS score for female was 75, and for males, 72.5. These scores are comparable and both above the average suggested by Sauro (2011).

Many of the constructs in the SUS measure intuitiveness, either directly or indirectly, so this above average score indicates that NFR3-Be Intuitive was successfully fulfilled.

6.1.2.4 Further Questions
6.1.2.4.1 Portability
The volunteers were asked what platform they used in the evaluation session, the results are shown in Table 6.7.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS X</td>
<td>Firefox</td>
</tr>
<tr>
<td>MacOS</td>
<td>Google Chrome</td>
</tr>
<tr>
<td>Windows</td>
<td>Google Chrome</td>
</tr>
<tr>
<td>MacOS High Sierra</td>
<td>Safari 11.1.1</td>
</tr>
<tr>
<td>Windows 10</td>
<td>Google Chrome</td>
</tr>
</tbody>
</table>

Table 6.7: Platform Used

Of the problems reported by the volunteers, all were perceived flaws inherent in the system, and not related to portability. Unfortunately, none of these volunteers used a mobile device to take part; however, cursory testing on Chrome for Android
Chapter 6 Results

suggests that compatibility, allowing the successful completion of a lecture. From the limited data available, NFR2-Be Portable appears to have been fulfilled.

6.1.2.4.2 Problems Encountered
One student encountered a problem where it becomes difficult to input data – if the area around the mutable area is selected and erased, the non-mutable characters remain, but the mutable area is reduced to a single character. Because the whole line is highlighted, not just the individual characters, it becomes difficult to locate the single point from which input is accepted, leading to student’s assuming that input is impossible.

A potential remedy would be to highlight only the mutable characters, so that a single green space would remain, making it clear that input is still possible and from where it is possible.

Another student had issues with one of the problems in which they were required to write 3 method bodies. Upon submitting 2 correct methods and one incorrect, they failed all test groups, as the class could not compile.

This issue was not unexpected. In the case of the problem above, the functionality of methods was not interdependent, and, as such, the problem should have been split into 3 problems. For more complicated problems in which the student must complete several interdependent sections of code, some means of substituting un compilable segments with sections of the model solution would rectify this issue. This could be done, for example, by allowing the lecturer to arbitrarily demarcate solution segments for the system to swap after a failed compilation attempt.

6.1.2.4.3 Suggestions Received
One volunteer suggested giving useful advice when a test was failed; this highlights a flaw in the evaluation session. The feedback String attached to test groups exists for this very purpose; however, distracted by smaller issues, I neglected to provide this, instead merely restating what the methods should do. The screenshot provided in Figure 5.6 of Section 5.7.1.2, shows how this feature is supposed to be utilised.

Another suggestion was to provide a live run-through of the code, showing the values of variables and the changes mad as code is executed line-by-line. This would be a complicated feature to implement and would not be priority for quick exercises in a lecture, it could prove useful in some contexts.

More detailed skeleton code was suggested, but in practice this would be the responsibility of the lecturer to provide, and this comment was made without the context of a lecture supporting the material presented.

Sound effects and audio for the visually impaired were suggested. If provided, it would have to be in such a way as to only play for those who require it, perhaps through earphones. This raises wider
questions of accessibility: Would certain colour schemes suite people with visual impairments? Can tactile feedback be provided? What other accessibility issues affect the system?

6.1.2.4 Liked Features
The volunteers voiced approval for the code highlighting, provided by CodeMirror, and the ability to pose anonymous questions.

6.2 Conformance to Specification
The tables below show the status of each requirement, whether it was successfully implemented, and a description of how it was implemented, or, if not, why not.

6.2.1 General Functional Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Status</th>
<th>Conformance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFR1-End on Complete</td>
<td>Yes</td>
<td>Each presentation of a lecture keeps a list of incomplete attempts and ends automatically when this becomes empty. A flaw exists in that if one student begins and completes an attempt before their classmates join, the lecture will end. This can be avoided by allowing several students to join before commencing the presentation or by checking <em>Keep open after last attempt?</em> in the lecture plan options.</td>
</tr>
<tr>
<td>GFR2-Anonymise</td>
<td>Yes</td>
<td>Students cannot see each other’s results and questions posed to the lecturer are anonymous; however, a compromise was made to accommodate LFR3-Provide Lecturer Summary; lecturers can see individual student submissions and results, linked to the student’s name. This allows the lecturer to keep track of the state of individual student’s knowledge.</td>
</tr>
<tr>
<td>GFR3-Evaluate Immediately</td>
<td>Yes</td>
<td>When the student clicks <em>Evaluation Solution</em>, their solution is compiled, unit tests are run and regex tests checked, with the result appearing in their browser.</td>
</tr>
<tr>
<td>GFR4-Generate URL</td>
<td>Yes</td>
<td>When the lecturer creates a new presentation of a lecture, a URL is automatically generated and presented to them.</td>
</tr>
<tr>
<td>GFR5-Evaluate Tests Immediately</td>
<td>No</td>
<td>Unit testing problems were not implemented in the system.</td>
</tr>
</tbody>
</table>

Table 6.8: General Functional Requirements Implementation

6.2.2 Student Functional Requirements
Requirements for students (SFR):

<table>
<thead>
<tr>
<th>ID</th>
<th>Status</th>
<th>Conformance</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFR1-Display Problems</td>
<td>Yes</td>
<td>Once the lecture starts, each student who has joined is shown the first problem. Upon submission of each problem, the student is presented with the next problem. Upon submission of the final problem, the student is presented with a holding screen, or, if the lecture is finished, their placement and scores.</td>
</tr>
<tr>
<td>SFR2-Display Next</td>
<td>Yes</td>
<td>As above.</td>
</tr>
<tr>
<td>SFR3-Give Student Solution Feedback</td>
<td>Yes</td>
<td>The results of GFR3-Evaluate Immediately are presented to the student upon submission.</td>
</tr>
<tr>
<td>SFR4-Allow Resubmit</td>
<td>Yes</td>
<td>When the student clicks <em>Evaluate Solution</em>, their solution is not immediately submitted. They have option of clicking Try Again or Submit Solution.</td>
</tr>
</tbody>
</table>
Chapter 6 Results

<table>
<thead>
<tr>
<th>SFR5-Provide Student Summary</th>
<th>Yes</th>
<th>At the end of the presentation, the student receives their score as a breakdown of their performance on each problem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFR6-Rank Student</td>
<td>Yes</td>
<td>At the end of the presentation the student is presented with a breakdown of their class’s performance, as a histogram, showing the distribution of their cohort’s scores.</td>
</tr>
<tr>
<td>SFR7-Pose Anonymous Questions</td>
<td>Yes</td>
<td>Above each problem is a button to allow the submission of anonymous questions. These are displayed to the lecturer linked only to the problem, with no names attached.</td>
</tr>
<tr>
<td>SFR8-Retain for Revision</td>
<td>No</td>
<td>This functionality was not implemented.</td>
</tr>
</tbody>
</table>

Table 6.9: Student Functional Requirements Implementation

6.2.3 Lecturer Functional Requirements

Requirements for lecturers (LFR):

<table>
<thead>
<tr>
<th>ID</th>
<th>Status</th>
<th>Conformance</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFR1-Commence Lecture</td>
<td>Yes</td>
<td>The lecturer can click Terminate Lecture on Moderate Lectures page, triggering a 60 second countdown, after which the lecture is terminated.</td>
</tr>
<tr>
<td>LFR2-Create Account</td>
<td>Yes</td>
<td>A registration page exists, which elicits a display name, password and email address. These are used to login.</td>
</tr>
<tr>
<td>LFR3-Provide Lecturer Summary</td>
<td>Yes</td>
<td>The lecturer can view the current progress of the students during the lecture and see the results at the end, by clicking View Progress on the Moderate Lectures page.</td>
</tr>
<tr>
<td>LFR4-Create Lectures</td>
<td>Yes</td>
<td>The lecturer can create lecture plans, from which they can instantiate lecture presentations.</td>
</tr>
<tr>
<td>LFR5-Add Problems</td>
<td>Yes</td>
<td>The lecturer can add new problems to a lecture plan.</td>
</tr>
<tr>
<td>LFR6-Specify Java Problem</td>
<td>Yes</td>
<td>When the lecturer creates a new problem, they provide a skeleton solution, a model solution, and a suite of unit tests. The model solution and unit tests are compiled using the Java compiler.</td>
</tr>
<tr>
<td>LFR7-Provide Java Problem Test Suite</td>
<td>Yes</td>
<td>As above.</td>
</tr>
<tr>
<td>LFR8-Trigger End</td>
<td>Yes</td>
<td>The lecturer may terminate a presentation of a lecture by clicking Terminate on the Moderate Lectures page.</td>
</tr>
<tr>
<td>LFR9-Specify End Time</td>
<td>No</td>
<td>This functionality was not implemented. The lecturer must manually terminate or wait for all student attempts to be completed.</td>
</tr>
<tr>
<td>LFR10-Provide Skeleton Notation</td>
<td>Yes</td>
<td>Yes. The lecturer provides a skeleton solution, which is presented to the student to complete. By default, no lines are editable. The lecturer specifies mutable lines using the tag: /<strong>mutable</strong>/.</td>
</tr>
<tr>
<td>LFR11-Demonstrate Solutions</td>
<td>No</td>
<td>This functionality was not implemented. Lecturers may, however, display solutions on the whiteboard manually.</td>
</tr>
<tr>
<td>LFR12-Add Regex Test to Java Problem</td>
<td>Yes</td>
<td>The lecturer may add a regular expression tests and assign these to test groups.</td>
</tr>
<tr>
<td>LFR12-Specify JUnit Problem</td>
<td>No</td>
<td>This functionality was not implemented.</td>
</tr>
<tr>
<td>LFR13-Retain for Review</td>
<td>Yes</td>
<td>The lecture presentation is persisted and can be accessed at any time.</td>
</tr>
</tbody>
</table>
Chapter 7 Recommendations for Further Work and Conclusions

6.2.4 Non-Functional Requirements

The non-functional requirements, along with the objectives they fulfil, are as follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFR1-Be Reliable</td>
<td>Yes</td>
<td>The system appears to be stable in a range of conditions.</td>
</tr>
<tr>
<td>NFR2-Be Portable</td>
<td>Yes</td>
<td>In as much as no major compatibility problems were experienced on the platforms evaluated, and a test run on Android Chrome raised no issues; however, more stringent testing should be carried out.</td>
</tr>
<tr>
<td>NFR3-Be Intuitive</td>
<td>Yes</td>
<td>The system performed satisfactorily in the usability evaluation.</td>
</tr>
<tr>
<td>NFR4-Be Responsive</td>
<td>Yes</td>
<td>The system compiles and evaluates solutions and in a timely manner and remains usable under extreme load.</td>
</tr>
<tr>
<td>NFR5-Be Extensible</td>
<td>Yes</td>
<td>Design patterns were used to produce a cohesive structure with low coupling.</td>
</tr>
</tbody>
</table>

Table 6.11: Non-Functional Requirements

Chapter 7

Recommendations for Further Work and Conclusions

7.1 Recommendations for Further Work

Whilst the evaluation of the objectives and hypotheses did not yield any statistically significant results, they did not reject the fulfilment of the objectives or the truth of the hypotheses. All but the evaluation for Hyp-Understanding had a positive average. No conclusions can be drawn, but these results do merit further investigation.

The system requires a larger evaluation as part of a real course, with real students. This evaluation should be designed to make the most of the features afforded by this system, with well-thought-out hints provided in the student feedback, and problems which draw on the material presented in the course. With a larger sample size, the results may have more statistical significance.

A further evaluation should be careful to explicitly ask students if they have a background in Java, as the general question on programming skills in the survey conducted here failed to elicit this information. It would be useful to evaluate if this prior-knowledge has any effect on the student’s perception of the system.

Currently, the entire solution fails all tests if it cannot be compiled. Some means of segmenting a problem, either by method or arbitrary point in the code, should be introduced, so failures in one area
of problem do not prevent the whole solution from being compiled and evaluated. Evaluation of invalid code could take place by methodically swapping out segments with segments from the model solution, failing the tests related to this segment, and attempting to compile.

Improving accessibility and investigating alternative means of input and feedback should be investigated, to ensure that no students are excluded from the activities. Inclusivity could also be aided by testing on a wider range of devices to ensure that those without laptops can take part.

If it was established that the system was of use, it would be beneficial to expand the scope of the system beyond Java and to other programming languages.

Deficits in student’s understanding of testing were discussed in the literature review, but not acted upon in the implementation, so the implementation of unit testing problems should still be pursued.

7.2 Conclusions

This project attempted to produce a student response system, with the purpose of improving learning outcomes by increasing engagement, encouraging interaction and allowing students to experiment with programming constructs, whilst receiving useful feedback.

If the objectives were to be measured purely by the fulfilment of requirements, they were generally achieved; however, the user evaluation did not produce any statistically significant results. This was due to the difficulty in finding volunteers when the pool of potential candidates was also busy with their own theses.

Since lecturers were not involved in the evaluation, Hyp-Feedback Lecturer, Obj-Augmentation, and Obj-Progress, were not formally assessed.

Feedback from the usability survey, however, indicated that the system exhibited above average fitness for purpose.

With these points in, I would conclude that the background research, and software development elements of the project were more successful than the evaluation.

Despite this lack of statistical significance, the responses gathered were on average positive. For this reason, the aims and objectives of this project are worth pursuing further. A large body of research supports the idea that student response systems improve learning outcomes through increased engagement.

I believe I allocated too much time to developing the software, trying to perfect the lecturer’s interface, which, on reflection, was less important to the overall goal as getting good research results. A simpler software solution, implementing the same student interface, evaluated earlier and with more subjects, and in a more realistic setting, would have been informative.
References


References


References


References


Appendix A

Lecturer’s Guide

A.1 Register and Login

To create and moderate lectures, you must first register login. When you first visit the application in a browser, you will see the following page:

From this page you may either register or login. To register, click register and complete in the following form:
Upon submission, your new account will be created, and you will be brought to the Moderate Lectures page. You may login to this account at any time by clicking login on the welcome page.

A.2 Creating and Modifying a Lecture Plan

From the Moderate Lectures page, you may click the New Lecture button to create a new lecture plan. A new lecture plan will be created and populated with default data.
From this page enter a new lecture title as well as a description and click save. The details will be stored. From here you may edit problems, delete them, and create new ones. Click edit problem to edit the default problem. By checking or unchecking *Keep open after last attempt?*, you can specify whether you want the lecture to remain open after the last student submits their attempt, or to close itself automatically after the last student finishes their attempt.

### A.3 Modifying a Problem

Upon clicking *Edit Problem* in the *Edit Lecture Plan* page you will be presented with the *Modify Problem*, as shown below.
In the example above, unit tests for a simple Java problem have been added. In this example, instructions for a test in which the student must return the absolute value of an argument using the ternary operator (condition ? arg : arg), have been supplied. You enter your own custom data in this page.

To create your own problem, at a minimum you need: the problem title, a class containing unit tests named SolutionTest, a solution class name Solution and a skeleton.

The SolutionTest class needs a matching Solution class to compile. The unit tests in the tests’ class should call on the methods of either an instance of Solution or its static methods. The method names of each method marked @Test in the unit tests’ file will be extracted and used to provide the names of the tests of in the problem.

The skeleton is class is the partially written solution provided to the student when they attempt the problem. You can provide as much detail as you like, along with comments and instructions for the student to fill in the gaps provided. By default, the whole skeleton is immutable – the student may not edit any part of it. By placing the tag /**mutable**/ at the end of the, the line becomes mutable – the student may input their own code into this line.
After you have provided this information, you may press submit, where you will be taken to the Edit Tests page.

In this example, three tests have been extracted from the unit tests’ class and added to the problem in the default test group. As the names of unit tests must match those in the unit tests’ class, you cannot edit these. You can, however, change which test group they are assigned to.

Test Groups govern how solutions are assessed and graded. Each test group passes only if all the tests assigned to them are passed by the student’s solution. Each test group has a given number of points awarded to it, along with a description and a feedback message. The description is a short title displayed to the student, and the feedback is displayed to the student when a test in the group fails.

In this example, the three tests listed are mapped to methods in the unit tests’ class, each annotated with @Test, and sharing the same name. All three of these tests are related, passing a different value to the student’s solution, ensuring the absolute value is correctly calculated; therefore, these tests belong to the same logical group.
A new Test Group can be created by clicking New, clicking the Edit button by the new test, then filling in the details on the Edit Test Group page. After submitting, you will be returned to the Edit Tests and Groups page where you can assign tests to the new group using the drop-downs by the tests at the top of the page.

To create a new test group, click new at the bottom of the Test Groups section. This will create a new test group. By clicking edit, you can set its attributes, as in Figure A.7.

Figure A.7: The Edit Test Group page
In this example, to test if the student has accomplished this task of returning the absolute value using the ternary operator \( \text{condition} \ ? \ \text{arg} \ : \ \text{arg} \), it is necessary to use regular expressions to test for other conditional statements and the use of the \textit{Math} library. This can be achieved by adding regex tests, which fail if the given regular expression is found in the solution.

To create a new regex test, click \textit{New Regex Test} at the top of the \textit{Edit Tests and Groups} page. The \textit{Edit Regex Page}, as in Figure A.8, is shown where you can enter a name for the test and the regular expression it uses. Upon completion, you will be returned to the Edit Tests and Groups page, from which you can assign these tests to test groups.

A potential pitfall is, if the regex is included in the skeleton solution’s comments, the test will always fail as these tester cannot detect comments. This can be avoided by carefully wording the hints in the skeleton.

![Edit Regex Test](image)

*Figure A.8: The Edit Test Group page*

Figure A.9 shows all the tests and test groups created for the problem. Using the drop-down boxes by the test groups, you can assign each test group a dependency. A test group can only pass if its dependency also passes. This is useful in this example, as it affords the constraint upon \textit{Use ternary} whereby it can only pass if \textit{Return absolute} also passes. This is useful because it means the student can be awarded half marks for returning the absolute value and then the rest of the mark for using the
correct operator. Without this constraint, the student could pass *Use ternary*, but fail *Return absolute*, simply by returning 0.

![Edit Tests and Groups](image)

**Figure A.9 The completed problem**

A.4 Moderating Lectures

Each lecture has a lecture plan, from which presentations can be spawned. These presentations represent an actual instance of the lecture, as carried out in the classroom. You can create new
Appendix A Lecturer’s Guide

presentations, start them, terminate them, and view their progress from the Moderate Lectures Page. The new lecture plan is shown in Figure A.10.

Upon creation, the lecture plan will have no presentations; a new presentation can be created by clicking Spawn Lecture Presentation. There are also buttons to edit the lecture, as described above; delete it; and duplicate it.

You must terminate all active presentations before deleting a lecture presentation. Terminating presentations is described later in this section. You cannot edit a plan after a presentation has started; however, you can duplicate the lecture and edit the duplicate, and, if desired, delete the original plan.

Alongside the presentations listed on the Moderate Lectures page, are options to start the presentation, start the presentation with simulated dummy students, delete the presentation, as well as the student URL. If the lecture is started with dummy students, several fake students will join the lecture and pass and fail test groups at random. To allow your students to join the presentation, you can share the Student URL with them. They will be prompted to provide their name and will then either be presented with a holding page, if the presentation has not started, or the first problem if it has.

Once you start the presentation, it will appear as follows in Figure A.11.

The lecture will terminate when all students have completed it, or when you click Terminate. Upon termination, the students will receive a warning informing them that they have 45 seconds to submit
their solution, and after 60 seconds the presentation will terminate. The 15 seconds grace period is to take account in potential delays in sending updates to the students.

A.5 Observing Lectures Presentations

By clicking View Progress on the Moderate Lectures page, you can see how your students are progressing through the presentation as in Figure A.11. The students can ask anonymous questions, and these are displayed in at the bottom of the page. You may view a student’s solution by clicking the link in the pass/fail box. Dummy attempts do not provide solutions.

![Figure A.11: The View Progress page](image-url)
Appendix B

Student’s Guide

B.1 Joining a Lecture

To join a lecture, simply navigate to the URL provided by your tutor; this will be in a form like the following: http://www.codehort.net/code/join?id=186. Once you have joined you will be asked to provide your name, as in Figure B.1. You lecturer will be able to see your name alongside your results; however, any questions you ask in the lecture will be anonymous.

![Figure B.1: The Join Lecture page](image)

B.2 Taking Part

Once you have joined the lecture, if the lecture has not yet begun, you will be presented with a holding page. Once you lecturer starts the lecturer, you will be presented with the first problem. The problems appear as in Figure B.2.
Appendix B  Student’s Guide

Return Absolute with Ternary Operator

Using the ? operator, complete the method to return the absolute value of x.

Using the ? operator, evaluate the value of x and return the absolute value.
If, x is negative, return -x. If x is 0 or positive, return x. Do not use the maths library or any other conditional statements.

You will be awarded:
5 points for returning the absolute value of x.
5 points for using the ternary operator?

```java
class Solution {
    /**
     * Returns the absolute value of x, using the ternary operator (?).
     * @param x
     * @return An integer representing the absolute value of x
     */
    public static int absolute(int x) {
        if(x < 0)
            return -x;
        return x;
    }
}
```

Compiled Successfully

You scored: 5/10

Passes  Failures  Not Attempted

<table>
<thead>
<tr>
<th>Test</th>
<th>Points</th>
<th>Your Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return absolute</td>
<td>5</td>
<td>pass</td>
</tr>
<tr>
<td>Use ternary</td>
<td>5</td>
<td>fail</td>
</tr>
</tbody>
</table>

public static int absolute(int x) should use the ternary operator to carry out its work

Figure B.2: Attempt Problem
Appendix B Student’s Guide

The problem instructions appear at the top of the page. Beneath this is the solution input box. Enter your solution here in the areas marked green.

Once you think you have solved the problem, click Evaluate Solution, to see if it is correct. The results of the evaluation will appear below, with your performance on each part of the problem shaded according to its result. If you fail on a part, a description of why will appear underneath. Click Try Again if you would like to amend your solution or Submit Solution if you are satisfied.

Your class’s progress displayed alongside your last solution’s results as coloured bars, as passed, failed and not yet attempted respectively.

B.3 Asking a Question

You can pose an anonymous question to your lecturer at any time by clicking Ask a Question on the problem page.

B.4 Completion – Seeing Your Results

Once you have completed the lecture, you will be shown the lecture summary page as in Figure B.3. Below your overall score, your placement is shown in a histogram, so you can see how you did and compare it with your class. Below this, your performance on each problem is shown, along with bars indicating that of the rest of your class.
Appendix B Student’s Guide

Figure B.3: Lecture Summary Page
Appendix C

Figures

C.1 Connecting to a Presentation

Figure C.1: Connecting to a Presentation
C.2 Sending a Solution for Evaluation

Figure C.2: Sending a Solution for Evaluation
Appendix D

User Survey

D.1  The Consent Form

<table>
<thead>
<tr>
<th>Consent Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>The purpose of this session is to evaluate an educational tool. As part of this, you will be asked to participate in a short example lecture, where you will be presented with a series of short Java problems and asked to solve them. Your lecture solutions and results, along with your answers to this survey will be stored and anonymised. By continuing, you confirm that you understand the nature of the task and that your results will be anonymised for use in the study and agree to this. Would you like to continue?</td>
</tr>
<tr>
<td>Yes</td>
</tr>
</tbody>
</table>

Table D.1: Consent Form

D.2  The Student Details Form

Those who answer Yes to Stu-Comp were directed to the Place of Study form, shown in table D.3, whilst those who answer No to Stu-Comp were directed to the Past Study form, shown in table D.4.

<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
<th>Input</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stu-Gender</td>
<td>What is your gender?</td>
<td>Free text</td>
<td>With enough data from each category, this would have been used to examine difference in perception of the system between genders.</td>
</tr>
<tr>
<td>Stu-Comp*</td>
<td>Are you currently studying computer science or related subject?</td>
<td>Yes/No</td>
<td>Combined with Stu-Prev to compare the perceptions of students have studied computer science with those who have not.</td>
</tr>
<tr>
<td>Stu-Score</td>
<td>What was your score in the lecture?</td>
<td>Number</td>
<td>To examine potential correlations between overall score and perception of the system.</td>
</tr>
</tbody>
</table>

Table D.2: Student Details Form (* denotes mandatory)
Appendix D User Survey

D.3 The Place of Study Form

<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
<th>Input</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stu-Watt*</td>
<td>Are you currently studying at Heriot-Watt?</td>
<td>Yes/No</td>
<td>With enough data from each category, would have been used to compare perceptions between Heriot-Watt computer science students and those from elsewhere.</td>
</tr>
<tr>
<td>Stu-Lang</td>
<td>Would you like to add any more details describing your experience in programming languages?</td>
<td>Free text</td>
<td>To look for general patterns between knowledge of programming languages and perceptions of the system.</td>
</tr>
</tbody>
</table>

Table D.3: Place of Study Form (* denotes mandatory)

D.4 The Past Study Form

<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
<th>Input</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stu-Prev*</td>
<td>Have you previously studied computer science or related subject?</td>
<td>Yes/No</td>
<td>Combined with Stu-Comp to compare the perceptions of students have studied computer science with those who have not.</td>
</tr>
<tr>
<td>Stu-Lang</td>
<td>Would you like to add any more details describing your experience in programming languages?</td>
<td>Free text</td>
<td>To look for general patterns between knowledge of programming languages and perceptions of the system.</td>
</tr>
</tbody>
</table>

Table D.4: The Past Study Form (* denotes mandatory)
D.5 The System Usability Scale (Brooke, 1996)

**System Usability Scale**


<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly disagree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. I think that I would like to use this system frequently</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>2. I found the system unnecessarily complex</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>3. I thought the system was easy to use</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>4. I think that I would need the support of a technical person to be able to use this system</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>5. I found the various functions in this system were well integrated</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>6. I thought there was too much inconsistency in this system</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>7. I would imagine that most people would learn to use this system very quickly</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>8. I found the system very cumbersome to use</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>9. I felt very confident using the system</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
<tr>
<td>10. I needed to learn a lot of things before I could get going with this system</td>
<td>1 2 3 4 5</td>
<td></td>
</tr>
</tbody>
</table>

*Figure D.1: The System Usability Scale, (Brooke, 1996).*
D.6 Objectives and Hypotheses Likert

<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sys-Engage-</td>
<td>I find other systems, such as Socrative, more engaging than this one</td>
<td>To evaluate Hyp-Engagement and the status of Obj-Engagement.</td>
</tr>
<tr>
<td>Sys-Understand-</td>
<td>I found the material presented in lectures where other systems, such as Socrative, were used easier to understand</td>
<td>To evaluate Hyp-Understanding.</td>
</tr>
<tr>
<td>Sys-Feedback-*</td>
<td>The feedback provided was of no use to my education</td>
<td>To evaluate Hyp-Feedback Student and the status of Obj-Feedback.</td>
</tr>
<tr>
<td>Sys-Feedback+*</td>
<td>The feedback provided by the system helped me in my efforts in learning the programming language</td>
<td>To evaluate Hyp-Feedback Student and the status of Obj-Feedback.</td>
</tr>
<tr>
<td>Sys-Understand+</td>
<td>I understood the material presented better than that presented in lectures where another system, such as Socrative, was used</td>
<td>To evaluate Hyp-Understanding.</td>
</tr>
<tr>
<td>Sys-Engage+</td>
<td>I felt more engaged with the material than in lectures where Socrative or another system was used</td>
<td>To evaluate Hyp-Engagement and the status of Obj-Engagement.</td>
</tr>
<tr>
<td>Sys-Interact Lecturer+*</td>
<td>This tool would encourage me to interact with my lecturer</td>
<td>To evaluate the status of Obj-Interaction.</td>
</tr>
<tr>
<td>Sys-Interact Peers+*</td>
<td>Sharing a device, this tool would encourage me to interact with my peers</td>
<td>To evaluate the status of Obj-Interaction.</td>
</tr>
<tr>
<td>Sys-Interact Lecturer-*</td>
<td>I would not be inclined to interact with my lecturer as a result of using this tool</td>
<td>To evaluate the status of Obj-Interaction.</td>
</tr>
<tr>
<td>Sys-Interact Peers-*</td>
<td>This tool would not encourage me to communicate with my peers, if we shared a device</td>
<td>To evaluate the status of Obj-Interaction.</td>
</tr>
</tbody>
</table>

Table D.5: Objectives and Hypotheses Likert (* denotes mandatory)

D.7 Final Comments – Free Text Questions

<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sys-Remove</td>
<td>Considering the user interface, do you have any comments on which features should be added, which should be kept and which should be removed?</td>
<td>General insight and eliciting recommendations for further work.</td>
</tr>
<tr>
<td>Sys-Platform</td>
<td>Which platform did you use to complete the problems (device/OS/browser)?</td>
<td>Used alongside Sys-Problems to evaluate the extent to which Obj-Portability has been achieved.</td>
</tr>
<tr>
<td>Sys-Problems</td>
<td>Did you experience any problems? If so, please describe them.</td>
<td>Used alongside Sys- Platform to evaluate the extent to which Obj-Portability has been achieved.</td>
</tr>
</tbody>
</table>

Table D.6: Final Comments – Free Text Questions (all fields are optional)
Appendix E

Evaluation Results

E.1 Load Test Results

<table>
<thead>
<tr>
<th>Attempt No</th>
<th>Wave</th>
<th>Time Taken (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1705</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>14035</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>14589</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2562</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6500</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1501</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>837</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>7162</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>815</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>38444</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>9719</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>10417</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>35393</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>11531</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1038</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>5278</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>13390</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>12613</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>8335</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>9063</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>37168</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>34228</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td>3182</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>3883</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>11989</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>25790</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
<td>10931</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>28077</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td>34434</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>28612</td>
</tr>
<tr>
<td>31</td>
<td>3</td>
<td>33646</td>
</tr>
<tr>
<td>32</td>
<td>3</td>
<td>32982</td>
</tr>
<tr>
<td>33</td>
<td>3</td>
<td>23777</td>
</tr>
<tr>
<td>34</td>
<td>3</td>
<td>27153</td>
</tr>
<tr>
<td>35</td>
<td>3</td>
<td>26509</td>
</tr>
</tbody>
</table>

Table E.1: Load Test Results
### Appendix E Evaluation Results

#### E.2 Usability Survey Results

<table>
<thead>
<tr>
<th>What is your gender? (This is optional)</th>
<th>Male</th>
<th>Male</th>
<th>Female</th>
<th>Male</th>
<th>Not Given</th>
<th>Female</th>
<th>Male</th>
</tr>
</thead>
<tbody>
<tr>
<td>I think that I would like to use this system frequently</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>I found the system unnecessarily complex</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>I thought the system was easy to use</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>I think that I would need the support of a technical person to be able to use this system</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>I found the various functions in this system were well integrated</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>I thought there was too much inconsistency in the system</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>I would imagine most people would learn to use this system very quickly</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>I found the system very cumbersome to use</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>I felt very confident using the system</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>I needed to learn a lot of things before I could get going with this system</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SUS Score</td>
<td>95</td>
<td>85</td>
<td>70</td>
<td>52.5</td>
<td>97.5</td>
<td>80</td>
<td>57.5</td>
</tr>
</tbody>
</table>

Table E.2: Usability Survey Results
### E.3 Hypotheses and Objectives Evaluation Raw Results

<table>
<thead>
<tr>
<th>Question</th>
<th>Female</th>
<th>Female</th>
<th>Male</th>
<th>Male</th>
<th>Male</th>
<th>Male</th>
<th>Not Provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>I find other systems, such as Socrative, more engaging than this one</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>N/A</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>I found the material presented in lectures where other systems, such as Socrative, were used easier to understand</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>N/A</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>The feedback provided was of no use to my education</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>The feedback provided by the system helped me in my efforts in learning the programming language</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>I understood the material presented better than that presented in lectures where another system, such as Socrative, was used</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>N/A</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>I felt more engaged with the material than in lectures where Socrative or another system was used</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>N/A</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>This tool would encourage me to interact with my lecturer</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Sharing a device, this tool would encourage me to interact with my peers</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>I would not be inclined to interact with my lecturer as a result of using this tool</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>This tool would not encourage me to communicate with my peers, if we shared a device</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

*Table E.3: Hypotheses and Objectives Evaluation Raw Results*