Spring 2013

Class control: An adaptable and self configuring classroom control system

Thomas Grimes
James Madison University

Follow this and additional works at: https://commons.lib.jmu.edu/master201019
Part of the Computer Sciences Commons

Recommended Citation
https://commons.lib.jmu.edu/master201019/226

This Thesis is brought to you for free and open access by the The Graduate School at JMU Scholarly Commons. It has been accepted for inclusion in Masters Theses by an authorized administrator of JMU Scholarly Commons. For more information, please contact dc_admin@jmu.edu.
Class Control: An Adaptable and Self Configuring Classroom Control System

Tom Grimes

A thesis submitted to the Graduate Faculty of JAMES MADISON UNIVERSITY in Partial Fulfilment of the Requirements for the degree of Master of Science

Computer Science

May 2013
Declaration

I declare that the material submitted for assessment is my work except where credit is explicitly given to others by citation or acknowledgement and abides by the JMU honor code.

The main text of this project report is 28547 words long. The whole paper is 34567 words long.

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

Tom Grimes
## Contents

Declaraton .......................................................................................................................... ii  
Contents ............................................................................................................................. iii  
Table of Figures ................................................................................................................... viii  
Table of Figures .................................................................................................................... x  
Abstract ............................................................................................................................... xi  

### 1. Introduction

1.1 Problem ......................................................................................................................... 1  
1.2 Project Methodology ..................................................................................................... 2  
1.3 Project Success ............................................................................................................. 3  
1.4 Report Outline .............................................................................................................. 3

### 2. Objectives

2.1 Primary Objectives ....................................................................................................... 5  
2.1.1 Self-Configuration ................................................................................................. 5  
2.1.2 Machine Learning ................................................................................................. 5  
2.1.3 Self-Adaptability ................................................................................................. 6  
2.2 Secondary Objectives ................................................................................................. 6  
2.2.1 Security .................................................................................................................. 6  
2.2.2 Load previous configurations ............................................................................... 6  
2.2 Tertiary Objectives ...................................................................................................... 7  
2.3.1 Audit Log ............................................................................................................... 7  
2.3.2 Distributed sub-systems in classrooms ................................................................ 7

### 3. Literature Review

3.1 Adaptation ..................................................................................................................... 8  
3.1.1 Adaptability design questions ............................................................................... 11  
3.1.2 A sub-system approach to adaptability ................................................................ 12  
3.1.3 The MUSIC project ............................................................................................. 14  
3.1.4 A Model Based Approach .................................................................................... 16  
3.1.5 Self Configuring Networks ................................................................................... 17  
3.2 Machine Learning ..................................................................................................... 20  
3.2.1 Machine learning to aid security ......................................................................... 21  
3.2.2 Machine learning for cognitive radio networks ................................................... 22  
3.2.3 Machine learning to tackle new problems ......................................................... 24  
3.2.4 Machine learning applied to oil spill detection ................................................... 24  
3.2.5 Unsupervised Learning ......................................................................................... 27  
3.2.6 Reinforcement Learning ......................................................................................... 27  
3.3 Evolutionary Programming ....................................................................................... 28

### 3.4 Network Discovery

3.4.1 Project Octopus .................................................................................................... 31  
3.4.1.1 Project Octopus ............................................................................................... 32
3.4.2 ENTRAPID Protocol Development Environment .............................................. 33
3.5 Existing Software.......................................................................................... 34
  3.5.1 AB Tutor Control................................................................................. 34
  3.5.2 Apple Remote Desktop ................................................................. 36
3.6 Conclusions................................................................................................. 36
4. Requirements Specification ............................................................................. 41
  4.1 Functional Overview ............................................................................. 41
    4.1.1 Teacher (Server).............................................................................. 41
      4.1.1.1 Finding Students (Clients) ......................................................... 41
      4.1.1.2 Gathering information about a classroom .............................. 43
      4.1.1.3 Letting the user choose a classroom to view ...................... 45
      4.1.1.4 Fault Tolerance ................................................................ 46
      4.1.1.5 Dynamic Screen Refresh Rate ................................................. 47
    4.1.2 Student (Client)................................................................................ 48
      4.1.2.1 Finding the teacher .................................................................. 48
      4.1.2.2 Establishing a connection ...................................................... 48
      4.1.2.3 Check if a server application opens on the same machine .... 49
  4.2 Structured Requirement Specifications ...................................................... 49
5. Software Engineering Process ........................................................................ 55
  5.1 Requirements Engineering ........................................................................ 55
  5.2 Methodologies Used ................................................................................ 55
    5.2.1 Prototyping ....................................................................................... 56
    5.2.2 Iterative Development ...................................................................... 57
  5.3 Timing ....................................................................................................... 58
  5.4 Weekly Meetings ...................................................................................... 59
  5.5 Coding Standards .................................................................................... 59
  5.6 Mercurial Repository ................................................................................ 60
  5.7 Testing ...................................................................................................... 60
6. Ethics ............................................................................................................. 62
7. Design ........................................................................................................... 64
  7.1 System Architecture ................................................................................ 64
    7.1.1 Teacher ............................................................................................. 64
    7.1.3 Model – View – Controller ............................................................ 65
      7.1.1.1 GUI .......................................................................................... 66
      7.1.1.2 Database .................................................................................. 66
      7.1.1.3 Network Commands ............................................................... 67
      7.1.1.4 Configuration File ................................................................. 67
      7.1.1.5 Audit Trail ................................................................................ 68
    7.1.2 Student ......................................................................................... 68
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.5.6 Saving Conversations</td>
<td>120</td>
</tr>
<tr>
<td>9.5.7 Security</td>
<td>120</td>
</tr>
<tr>
<td>10. Conclusions</td>
<td>121</td>
</tr>
<tr>
<td>10.1 Achievements</td>
<td>121</td>
</tr>
<tr>
<td>10.1.2 Extending the Original Project</td>
<td>121</td>
</tr>
<tr>
<td>10.1.3 Machine Learning</td>
<td>121</td>
</tr>
<tr>
<td>10.1.4 Self Configuration</td>
<td>122</td>
</tr>
<tr>
<td>10.2 Lessons Learned</td>
<td>122</td>
</tr>
<tr>
<td>10.3 Personal Summary</td>
<td>122</td>
</tr>
<tr>
<td>Appendices</td>
<td>124</td>
</tr>
<tr>
<td>Appendix A – Testing Summary</td>
<td>124</td>
</tr>
<tr>
<td>Appendix B – User Manual</td>
<td>136</td>
</tr>
<tr>
<td>Appendix C – Status Report</td>
<td>141</td>
</tr>
<tr>
<td>Appendix D - Interoperability</td>
<td>143</td>
</tr>
<tr>
<td>Bibliography</td>
<td>144</td>
</tr>
</tbody>
</table>
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Class Control in use in St Andrews</td>
<td>1</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Overall view of an EVM System</td>
<td>12</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Machines working together to solve a formula</td>
<td>13</td>
</tr>
<tr>
<td>Figure 4</td>
<td>MUSIC meta-model</td>
<td>15</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Architecture of a Self-Configuring Network</td>
<td>18</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Constraint Definition Language Example</td>
<td>19</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Self-optimization of cognitive user devices</td>
<td>23</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Steps of a Genetic Algorithm</td>
<td>30</td>
</tr>
<tr>
<td>Figure 9</td>
<td>AB Tutor Installation</td>
<td>35</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Simplified Listening for Clients</td>
<td>42</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Full Scan Process</td>
<td>43</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Simplified Client Scan while Main Software is Running</td>
<td>43</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Connecting to a Classroom</td>
<td>46</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Detecting Lost Clients</td>
<td>47</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Client Checking Server’s Status</td>
<td>47</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Connecting/Disconnecting to the Server</td>
<td>48</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Prototyping Methodology</td>
<td>56</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Scrum Framework</td>
<td>58</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Gantt Work Flow Chart</td>
<td>59</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Work Flow Chart Key</td>
<td>59</td>
</tr>
<tr>
<td>Figure 21</td>
<td>System Component Diagram</td>
<td>64</td>
</tr>
<tr>
<td>Figure 22</td>
<td>MVC Architecture</td>
<td>65</td>
</tr>
<tr>
<td>Figure 23</td>
<td>State Diagram of Client</td>
<td>69</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Communication Threads &amp; Classes</td>
<td>71</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Properties File</td>
<td>73</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Thumbnail View Screenshot</td>
<td>77</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Client’s Thumbnail Screenshot</td>
<td>78</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Broadcast Panel Screenshot</td>
<td>78</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Old Class Control Life Cycle</td>
<td>79</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Class Control Life Cycle</td>
<td>79</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Class Control Old Package Tree</td>
<td>81</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Class Control Interoperability Screenshot</td>
<td>83</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Client Added to the Database</td>
<td>85</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Database Tree</td>
<td>85</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Adding a Client to the Database</td>
<td>86</td>
</tr>
<tr>
<td>Figure 36</td>
<td>New Configuration Classes</td>
<td>87</td>
</tr>
<tr>
<td>Figure 37</td>
<td>Configuration Selection Screenshot</td>
<td>87</td>
</tr>
<tr>
<td>Figure 38</td>
<td>Configuration Selection 1</td>
<td>91</td>
</tr>
<tr>
<td>Figure 39</td>
<td>Configuration Selection 2</td>
<td>91</td>
</tr>
<tr>
<td>Figure 40</td>
<td>Broadcast Control Panel</td>
<td>92</td>
</tr>
<tr>
<td>Figure 41</td>
<td>Classroom Selection</td>
<td>94</td>
</tr>
<tr>
<td>Figure 42</td>
<td>Selecting Classrooms from Database</td>
<td>94</td>
</tr>
<tr>
<td>Figure 43</td>
<td>Edit Classroom Window</td>
<td>95</td>
</tr>
<tr>
<td>Figure 44</td>
<td>Edit Classroom Window 2</td>
<td>95</td>
</tr>
<tr>
<td>Figure 45</td>
<td>Initialization Messages</td>
<td>96</td>
</tr>
<tr>
<td>Figure 46</td>
<td>Broadcasting Output</td>
<td>99</td>
</tr>
<tr>
<td>Figure 47</td>
<td>Multicast Setup</td>
<td>101</td>
</tr>
<tr>
<td>Figure 48</td>
<td>Properties File with Port Numbers</td>
<td>102</td>
</tr>
<tr>
<td>Figure 49</td>
<td>XML Schema</td>
<td>104</td>
</tr>
<tr>
<td>Figure 50</td>
<td>Simplified XML Schema</td>
<td>105</td>
</tr>
</tbody>
</table>
Figure 52: Properties File Screenshot ................................................................. 106
Figure 53: Error Handling Client Loss ............................................................... 107
Figure 54: Remote Control Calculation ............................................................. 108
Figure 55: X Coordinate Calculation ............................................................... 109
Figure 56: Y Coordinate Calculation ............................................................... 109
Figure 57: Class Control Icons ....................................................................... 110
Figure 58: Password Entry Screen ................................................................. 136
Figure 59: Class Control Loading Screen ....................................................... 136
Figure 60: Scan or Load Panel .................................................................... 137
Figure 61: Configuration Selection ............................................................... 137
Figure 62: Classroom Selection ...................................................................... 138
Figure 63: Classroom Editing Screen 1 ......................................................... 138
Figure 64: Classroom Editing Screen 2 ......................................................... 139
Figure 65: Main Program ............................................................................ 139
Figure 66: Closing Screen ........................................................................... 140
Table of Figures

Table 1: Comparison of Final Project to Initial Requirements............................................117
Abstract

_classControl_ is a tool that was created at the University of St Andrews to allow a teacher to not only view all of the screens in a classroom at once but also take control of them and broadcast the teacher’s screen to them. Existing software wouldn’t allow the scalability to meet the demands of the size of the classroom so _classControl_ was developed and is still used by the school today.

However, this tool was created to be used in one classroom just running the Mac OS. This thesis project aimed to add the ability to move this tool out of the classroom and be able to configure itself in a new classroom containing machines with other operating systems that can run Java.

_classControl_ has now been extended to learn about a new environment it is in and use this information to help improve future runs of the tool. Previous configurations can be stored permitting a teacher to select a configuration that suits their needs and _classControl_ can also recommend a configuration depending on what clients it sees in the environment. _classControl_ can also be used to view and control Linux, Windows and Mac OS at the same time.
1. Introduction

This chapter of the report aims to give an initial overview of this thesis project. This section should give the reader a summary of the initial problem, how a solution was reached and an overview of the success of the project. This chapter ends with a brief look at the contents of this report.

1.1 Problem

*Class Control* is a tool that was created by the author of this project to allow a teacher to broadcast a presentation to a classroom of machines running Mac OS X. The software was then extended to allow the teacher to view screenshots of each of the machines in the classroom to see what the students were doing on them. The teacher is also able to use *Class Control* to take control of each machine individually and to also send text chat messages. The software was created as part of an undergraduate dissertation project, for use in a lab at St Andrews University, Scotland. Figure 1 shows a photograph of *Class Control* being used to display a presentation on each of the screens in the lab.

![Class Control in use in St Andrews](image)

*Figure 1: Class Control in use in St Andrews*

*Class Control* was originally intended to just be installed in the St Andrews lab; the goal of this project is to enable the software to work in James Madison University and
other schools and labs with minimal configuration from the person installing the software. This means that the machines will need to be able to discover each other in a classroom and be able to exchange information to allow the teacher’s machine to collect information about the student’s machines. The teacher’s application will have to store information about a classroom that it receives during the discovery phase so it can be used again in later sessions. Currently, the teacher’s application has a list of machine addresses hard coded into it. This project will add the functionality to *Class Control* to allow the teacher’s application to discover what machines there are in the classroom.

The main objectives of this project, therefore, are:

- To create a self-configuring piece of software that can discover other machines in the room that are using the *Class Control* software.
- To extend the software so that it will be able to store classroom configurations to save having to scan the environment each time it is run.
- To keep the original functionality of *Class Control* after making these changes.

**1.2 Project Methodology**

The process of developing a solution to the problem outlined in section 1.1 includes several key phases.

**Literature Review:** This phase will look at previous work in related areas to gain some further understanding of the problem and look at different ways of creating a solution.

**Design:** After gaining some inspiration from the literature review, the design process will look at how to tackle the problem and incorporate a solution into the existing *Class Control* software.
Implementation: This phase is where the design will be implemented and incorporated into Class Control. The report will explain how this has been done.

Evaluation: The final phase of the project will be an evaluation on how well the objectives have been met. There will also be a discussion on future work that could be done to further increase Class Control’s success and functionality.

These phases were completed using various software engineering techniques explained in chapter 5. These processes are needed to ensure that the project is completed on time and to ensure errors are minimized in the implementation phase.

1.3 Project Success

Class Control is a complete system that meets all of the primary and secondary objectives. One of the tertiary objectives was left out due to time constraints but does not affect the main goal of this project. The main goal of this project is to create self-configuring tool that can discover clients and learn some information about its environment. This main goal has been achieved and the project has been a success.

1.4 Report Outline

The report describes the development of this project over the year and shows the rationale behind different design ideas and software engineering techniques. The report will be split up into 10 chapters with relevant appendices attached to the end:

1. Introduction
   A discussion of what this project is about, its success and the content of this report

2. Objectives
   The goals of the project broken up into various categories showing their order of importance

3. Literature Review
A review of significant work that has already been carried out in the fields of study that this project relates to

4. Requirements

The requirements document shows the various criteria that the final system must meet

5. Software Engineering Process

Contains a description of how the project was developed and a review of the software engineering processes that were used.

6. Ethics

The ethical concerns that were raised in the development of this project and how they were addressed

7. Design

The design ideas of the main components and protocols in the software and also how the new changes can be tested

8. Implementation

A description of how the main components were implemented and how they function

9. Evaluation and Critical Appraisal

An evaluation of the software, comparing it to objectives and existing software with a discussion on future extensions to the software

10. Conclusions

A summary of the project’s achievements and the lessons learned throughout its development
2. Objectives

This section describes the project objectives established at the beginning of the project. The primary objectives are the main desired features of this program which, if not implemented, will result in this project being a failure and will prevent the lower priority objectives being completed. Secondary objectives are considered to be important and should be completed by the end of this project. The tertiary objectives are optional and are set as extension work, time permitting.

2.1 Primary Objectives

2.1.1 Self-Configuration

*Class Control* should be able to configure itself in a class room with minimal input from the class administrator. The software will be able to discover what clients are in the classroom and find out information about the machines such as their location and screen size. It needs to do this so that it can be installed in new environments as it is currently hard coded for the lab in St Andrews where it was developed.

2.1.2 Machine Learning

*Class Control* should be able to learn about its environment in order to help speed up future uses of the software. When *Class Control* learns the layout of a classroom, this information should be available the next time *Class Control* is run. The layout of the classroom would include the IP addresses of machines in the room and their screen sizes. The server would also discover if the classroom is arranged in a grid or not so that the clients can be displayed to the teacher in a way that resembles their physical position in the room. This would have to be entered by a lab administrator during the installation of *Class Control*. 
2.1.3 Self-Adaptability

The GUI in *Class Control* will have to reflect changes in the environment to the teacher and also ensure that a client leaving doesn’t affect the operation of *Class Control*. Throughout the runtime of *Class Control* it is likely that clients will leave and join a classroom. When this happens, the new client will have to be added to or removed from the teacher’s GUI. This will happen to make sure the teacher doesn’t try to communicate with a client that isn’t there or to allow the teacher to immediately communicate with the new client. The idea of clients leaving and joining the session can be thought of as a teacher starting a lecture and students arriving late or leaving early.

2.2 Secondary Objectives

2.2.1 Security

Security measures will have to be implemented to help prevent unauthorized connections to clients. The risk here is that an adversary could take advantage of a running client and pretend to be a teacher. This should not be allowed to happen because of the ability to view what is on a student’s screen and the ability to take remote control of the student’s machine. An audit log will also be provided to show connection details.

2.2.2 Load previous configurations

Objective 2.1.2 states that *Class Control* should remember any configuration that it learns so that the information can be reused the next time the software is run. *Class Control* will also be able to load not just the last known good configuration, but multiple configurations it has saved from previous runs of the software.

*Class Control* should also be able to suggest a configuration to the user based on what it learns from an initial scan of the environment. For example, it
could offer a configuration containing the most machines it sees from the initial scan.

2.2 Tertiary Objectives

2.3.1 Audit Log

Objective 2.2.1 mentioned the addition of an audit log to Class Control. This would be a log file of any connections made between the clients and the server and also when remote control connections are made. The audit log would contain IP addresses involved in the connection and also what time the connection happened. This could be used to help administrators work out who was using the machine at the time of an event happening to help with an investigation. The audit log could also log chat conversations between the teacher and the students.

2.3.2 Distributed sub-systems in classrooms

If the teacher has the capability of receiving messages from any client in any classroom, this could result in more messages than the teacher could handle. A single client in a classroom could be elected as the class leader and be responsible for telling the teacher’s application what machines are present in that room. Meeting this objective would help ease this burden. This could result in a lot more work than expected so this has been set as a tertiary objective.
3. Literature Review

The literature review portion of this report will look into existing research and software to look at how various features of the intended final project are already being implemented. This section will also help with the design process of this project as papers will inspire new ideas and reveal the most up to date methods of solving the various problems of this thesis. The main fields that this section will look at, with the rationale for choosing these fields, are:

**Adaptation:** *Class Control* will be running in a new and changing environment. It will need to adapt to changes such as clients leaving and joining a classroom. This section will look at how some existing systems approach adaptability in the environments they run in.

**Machine Learning:** *Class Control* will potentially be running across multiple classrooms and so will be running in a more complicated and, possibly, a much larger environment. Because of this *Class Control* will need to learn what it discovers about an environment each time it is run and make suggestions to the teacher about which configuration to use.

**Evolutionary Programming:** Learning from previously discovered information and building up a database should help the program keep evolving so that it always uses the most up to date information that it has gathered. Evolutionary programming may help uncover some algorithms or new concepts to achieve this.

**Network Discovery:** For the added functionality of adaptation and self-configuration there will have to be a new component in the software that allows the server to discover which clients are running in the same environment.

**Existing Software:** *Class Control* was originally built from inspiration from other software with similar functionality. This section will take another look at these pieces of software to see how they are installed in a classroom and see if *Class Control* could take any more inspiration from their approach.
3.1 Adaptation

A formal definition for adaptation would be (Subramanian, 2001) that adaptation means a ‘change in the system to accommodate change in its environment. More specifically, adaptation of a software system is caused by change from an old environment to a new environment and results in a new system that ideally meets the needs of its new environment’. Subramanian also states that adaptation involves three main tasks, given below with an example for each:

- **The system must be able to notice a change in the environment.**
  
  *This could be a node crashing suddenly or a new node joining the environment.*

- **The system must know what change to make to adapt to the environment change.** If a node crashes, then future attempts to establish a connection must be stopped to prevent an error. If a new node joins the environment, a connection must be established so it can be communicated with.

- **The system must have ability to make the changes needed to adapt to the change.** When the system has decided if a node has joined or left, it must make the changes needed to start or stop a connection.

A networked computer system that is built up of multiple distributed agents working together to cooperate and perform tasks is called a distributed system (Farley, 1998). There are a number of reasons why one would choose to create a distributed system. Farley describes three of these:

- Several small and cheap computers can be used to break down a task and solve it instead of using one expensive supercomputer
- Large data sets are hard to move around, distributed systems can be used to administer and view them remotely
• Redundant agents can be set up so that if a machine goes down, fault tolerance allows the task to still be carried out.

*Class Control* is essentially a distributed system that consists of a number of connected systems that rely on each other and make assumptions on the services that the other machines provide. One fundamental criterion to satisfy in the creation of these types of systems is to ‘embed the capability of recovery from unforeseen perturbances’ (Flatebo, 1991). Flatebo discusses how a system’s lifetime can be thought of as being in safe and unsafe states. A system could move into an unsafe state from a node unexpectedly disappearing, for example, and, thus, preventing communication with that node. Adaptation is the process in which a system takes steps to move from an unsafe state into a safe one as quickly as possible. The changes in an environment that could cause a system to adapt to its surroundings can be grouped into three categories (Reinhard, 2005):

- **Inter-individual** differences are where a piece of software can adapt to a user’s personal requirements such as language and font size etc. This also includes adapting to a user’s disability or personal characteristics such as emotion.

- **Intra-individual** differences consider the development of a single user throughout the lifetime of the software. Examples of this could be starting with a tutorial and adding extra levels of functionality as a user becomes more experienced.

- **Environment** differences come from dynamic computing environments. Software in this kind of environment has to tolerate changes in resources such as network connectivity or the number of clients being served.

Adaptability is also defined as (Oreizy, 1999) ‘self-adaptive software [that] modifies its own behavior in response to changes in its operating environment’. This definition is almost the same as the previous one. However, a slightly different definition from
Oreizy in a different paper gives the definition, ‘Adaptive evolution changes the software to run in a new environment’. This is a good definition for this project as the basic goal of this project to change the software so that it can work, not only in St Andrews, but also in James Madison University.

Software adaptability is very important to this project as the software won’t always be run in the same conditions. It may not always be run in the same classroom, and each classroom could have a different number of students logged in to machines at any time. This section of the literature review will look at various techniques in which previous work and research has already been carried out to develop adaptable systems.

### 3.1.1 Adaptability design questions

There are several questions that will need to be answered when designing a self-adaptive software system (Oreizy, 1999):

1. **What conditions does the system undergo adaptation?** The system could encounter node failure or want to improve response time.
2. **Should the system be open-adaptive or closed-adaptive?** Open adaptive means that the system would be able to apply new behaviors during runtime. Closed adaptive systems would make changes happen for the next runtime.
3. **What type of autonomy must be supported?** Is the system going to be fully automatic, run by a human or a mixture of the two?
4. **Under what circumstances is adaptation cost-effective?** The change made must outweigh the costs of not making the change, for example, is the change going to improve performance or hinder it.
5. **How often is adaptation considered?** Is there going to be a scan every second, or every five seconds or more to determine if a change is needed
6. *What kind of information must be collected to make adaptation decisions?* Is information going to be approximated or are precise readings going to be read to determine what decisions need to be made.

### 3.1.2 A sub-system approach to adaptability

One idea for creating a self-adapting system is to create a series of sub-systems that can work together to solve one larger problem (Nowostawski, 2005). The author discusses how evolutionary programming techniques at the time were not particularly useful in dynamic environments. This was because algorithms that were based on random mutation and selection were inefficient as they were built on a single system. Evolutionary algorithms will be explained later in this literature review but, put simply, they allow a set to of data to ‘mutate’ in the hope that one or more the mutations will be closer to an answer. The Evolvable Virtual Machine (EVM) created by Nowostawski uses traditional techniques but in a novel framework consisting of multiple cells that can change their functionality to meet changes in their environment. The sub-systems can specialize in different tasks and interact with their neighbors for either more processing power or to work together on assignments. This system creates a web of interconnected machines.

![Figure 2: Overall view of an EVM System](image)

Nowostawski argues that some random mutation combined with specialization is essential for a more complex system that can be more
efficient for larger problems. He also states that learned information from the past should also be stored to help with future problems and current evolutionary algorithms, at that time, didn’t do this. Because of this, current algorithms, at the time, couldn’t be used as effectively in an incremental self-improvement fashion.

The paper goes on to discuss how a web of interacting agents can cooperate with each other and, as a team, solve larger problems.

Figure 3: Machines working together to solve a formula

Figure 3 shows an example from the paper of a run of an algorithm across 20,000 iterations. The key at the top shows what each of the shades of gray represent. The final goal is to solve $3x+2y$, the darkest shade of gray in the image. The middle grid shows that after 10,000 iterations the machine at (1,3) has used its neighbor to the left with the answer to $2y$ and the neighbor below with the answer to $3x$ to create the answer to $3x+2y$. By splitting up a problem, machines can work on smaller problems to help solve larger problems.

This is linked to adaptability because neighboring machines can detect when neighboring machines have a solution that they can use. This is mentioned here to be linked to the possibility of clients running *Class Control* being able to figure out their coordinates in a lab environment if an admin user has only entered information about some of the clients. For example if a client knows
it is next to one machine and below another it can then tell the teacher’s application exactly where it is without having been told in the first place.

3.1.3 The MUSIC project

MUSIC is a self-adapting project that looks at adapting its use by learning about its environment (Rouvoy, 2009). This project is aimed towards a ubiquitous computing environment where several unexpected changes can happen throughout the execution of the software. Software using the MUSIC framework will contain various components that can plug in to the system depending on the execution context. This can help optimize the application and make it more effective in different situations. An example of this could be a cell phone that is frequently roaming and having to adapt to new network conditions. These changes not only require the cell phone to provide certain features throughout the network changes but the paper also considers the fact that a satisfactory Quality of Service also needs to be maintained. Applications need to constantly look out for new services available on the network or detect losing a service and be able to choose alternatives, when available. This is costly and hard to implement. The MUSIC framework aims to alleviate this problem for the application creators and delegate most of the self-adaptability process to the generic middleware. This is done by relying on an application to tell the middleware what sort of adaptation capabilities the application has to perform so that the middleware ensures that these capabilities can be carried out by the application.

To select methods in to perform the adaptation of a MUSIC application (Figure 4), a configuration of Component Realizations is deployed which provides the best utility to the application. This utility is a measurement of how fulfilled a user’s requirements will be met with the change, while optimizing resource utilization.
Plans are made to enable the middleware to make decisions on what configurations to use in different contexts. The adaptation reasoner builds a list of valid configurations while discarding configurations that can’t currently be used. Heuristics are then used to help determine which configuration to switch to, if a switch is needed.

To successfully meet the requirements of an adapting system like this the paper states four requirements need to be met:

1. Dynamic discovery of services
2. Dynamic binding and change of binding to service providers
3. Negotiation of service level agreements and detection of violations
4. Hosting and publishing of services

The discovery of services is handled by a set of discovery protocols. These are used by the MUSIC platform to build up information about the service capabilities and QoS requirements. This information is converted into plans which can be used to determine what choices are to be taken when an adaptability decision needs to be made. Service level agreements are constantly monitored to make sure they are being met. When the MUSIC middleware architecture detects an agreement isn’t being met then this
service is terminated and the adaptation process begins looking for a new
service to replace the terminated one.

3.1.4 A Model Based Approach

Embedded information systems show a need for a self-adaptive architecture
(Karsai, 1999). These systems need to be fault tolerant, autonomous and
adaptive to environmental changes while providing an acceptable level of
performance. This paper was written because, at the time, the latest methods
of creating software were to exhaust all possible inputs and changes possible
in the given environment. This led to unmanageable projects that took a long
time to design and debug. Adaptive systems combat this by providing
feedback that constantly monitors the system and can change how it functions
accordingly.

A model based approach to design an adaptive piece of software allows the
designers to create an abstract view of the software and its capabilities. This
helps make the project more manageable as aspects of the model can be
implemented separately. Using a model based approach to this project would
help establish three key areas of the project. The first is the design of the
project, the protocols, how the components will interact and learn about each
other and also how the software can be evaluated and tested. The second is
looking at transforming these models into an executable piece of software
and the third is using the software to carry out an analysis of the software’s
performance and to carry out tests to ensure the project is functioning
correctly. These ideas are very important to the core software engineering
principles of this project to help create a robust piece of software that can be
tested and evaluated.
3.1.5 Self Configuring Networks

In creating a tool, NESTOR (Konstantinou, 2002), to help automate the configuration of a network the author illustrates that there are several factors that make self-configuring networks challenging. These are:

The change propagation problem: This problem involves changes in multiple elements in the network. The software needs to:

- Recognize the various elements and their states i.e the network topology
- Represent the sequence of changes in these elements i.e change propagation rules
- Make the needed changes in various elements and handle unexpected events
- Enable recovery and undoing of changes in case they cause failures

The configuration policy problem: Configuration changes may cause erroneous states, resulting in software failure. To handle this problem, a piece of software will have to:

- Represent policy knowledge to know how to maintain a valid state
- Enforce these policy restraints
- Allow organizations to program their own policy constraints to meet their own policies

The composition problem: A self-configuring network needs to be built on an architecture that can solve the previous two problems. The paper suggests creating a hierarchy of components shown in the figure below.
The various components will able be to interact with each other to maintain the policies and safe states while handling and making changes to the system. An application will have the ability to use the self-configuring layer. The self-configuring management layer will apply the policy rules to the network. It will be able to use information that the modeling layer learns about the network elements.

NESTOR was created to meet these challenges and automate the configuration of a network to help reduce the complexity, time and cost of administrators. The tool is able to maintain system integrity and recover from failed nodes. The first step in NESTOR’s creation was to create a method of modeling the configuration information of the network. This was a difficult process because of the various network elements and various ways that they store configuration information. By creating a component using the Resource Definition Language (RDL) that can group together common elements and access common features, NESTOR is able to create a model of the network allowing easy access to information about nodes and saving administrators having to look at a variety of manuals to figure out a more laborious approach to gathering the same data.

Using this model, the next step is for NESTOR is model the semantics of the network. The Constraint Definition Language (CDL) is able to state assertions on objects in the RDL. An example of asserting a constraint is shown in the figure below.
This example will grab the hosts in the system and ensure that any hosts without a null name will all have a different name. There is also a similar Policy Definition Language to create policy rules.

NESTORS architecture is similar to that in figure 5 but is explained here:

NESTOR applications will access the repository through a Directory Access Protocol. It uses protocol proxies to communicate with various servers or elements such as DHCP. If a request is made, there is a check first to see if the transaction will cause a policy violation before handing the result to the host calling the DHCP. The Directory Management Protocol is used by Directory Servers to maintain the distribution and caching of resource objects. Distributing these objects allows the repository to scale well and be accessed quicker in different locations.

The Self-Configuration Management Layer is able authorize any changes that are made in the model to maintain a valid state. It prevents any model configuration changes from happening unless all of the policy rules have been evaluated and passed.

The Configuration Modeling Layer maintains a model of the network and stores network objects in a Resource Directory Server. This repository allows access to configuration settings and meta information about the network elements. If there are elements that aren’t NESTOR-enabled then a Protocol Adapter Layer provides the ability to pass information between the repository and the element.
NESTOR has been created to help reduce the manual effort needed by an administrator to handle the ever growing number of elements in a network. As networks get larger, the changes in them can happen more frequently and perhaps too fast for an administrator to maintain. NESTOR is a software based approach to automating the handling of these changes and maintaining a system stays in a valid state throughout its operation.

3.2 Machine Learning

For a system to become adaptable it will need to learn about its environment. Machine learning allows a system to modify itself, resulting in performance improvement on subsequent data, in other words, improve automatically with experience (Mitchell, 1997). According to Mitchell, ‘a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. Examples of successful applications of machine learning given by Mitchell include:

- Recognizing spoken words: Systems are able to recognize primitive sounds from captured speech and use a neural network to not only interpret these sounds but also customize itself for individual speakers.

- Learning to drive: A sensor-based system was able to successfully control and navigate a car at 70mph for 90 miles on public roads.

- Classification of new astronomical structures: Decision tree learning algorithms allows a system to interact with a large database to discover regularities and classify objects. In this example, NASA is able to use this system to classify celestial objects.

- Learning to play backgammon: By performing over a million practice games, a computer was able to foresee potential winning strategies and make moves that could help it win against a human opponent.
Techniques used in this system can be used in problems involving large search spaces.

Training mechanisms are applied to each of these applications so that a machine can initially be trained and then learn from this data. This allows the system’s future decisions in understanding speech or making a speed adjustment while driving a car will be the correct decision to make.

For this project, machine learning will allow the software to learn about its environment and use this information to enhance the performance of future runs. To learn about its environment, the system will have to communicate with other nodes in the network to learn where they are and how many machines are available to connect to. A protocol can be set up to aid with communication and some form of database will have to be set up to store and update information.

This section is closely related to adaptability, I have already mentioned how a system could use sub-systems or nodes to help break up the amount of learning the main program would have to perform. If a node disappears or a new node joins the network then the system will have to adapt to this. The main program will have a knowledge base containing information it learns from each sub system and subsequent machines it connects to. Each subsystem would also contain a knowledge base providing information about the classroom it is in so the teacher application doesn’t have to query each individual computer. This is particularly useful for a larger classroom size so that the teacher doesn’t have to query every machine in the school to find out which ones are in a specific classroom. Essentially, any adaptability decisions that are made will have to be stored so that the system can learn from them and use this new information in future runs.

3.2.1 Machine learning to aid security

Machine learning can help improve the security of the software. (Chan, 2006) talks about how machine learning algorithms can help improve the security of
a computer system. The paper discusses how much human effort has to be involved with detecting and extracting threats and how they think a ML algorithm could help speed up this process. One method in which ML is used to detect threats is by learning the difference between spam email and normal email. As new types of spam email are detected, the system can learn about these and use this information to check for other types of spam email. Another way spam email can be detected is by looking at text embedded in images, instead of in normal text form. As spammers use increasingly complex techniques to disguise text in images, an ML algorithm can learn about new patterns and adapt its search for text accordingly. These methods aren’t perfect but they show machine learning can provide some assistance in detecting and extracting threats before they become malicious.

3.2.2 Machine learning for cognitive radio networks

(Stavroulaki, 2012) states that ‘Learning mechanisms are essential for the attainment of experience and knowledge in cognitive radio systems, exposed to high dynamics with often unpredictable states’. A cognitive radio network is one where devices are able to choose from multiple wireless channels and choose the best one that gives the highest level of service. Learning mechanisms allows the system to learn about its current state and network configuration. Knowledge gained through these mechanisms can be used to quickly load configurations that have been known to be efficient in the past because they won’t have to relearn the information again. Knowledge in these systems can also be shared amongst nodes so that each node can also make their own more optimal decisions when communicating with the network.
Figure 7: Self-optimization of cognitive user devices

Figure 7 shows how a system acquires as much information as possible from the user and context before using this information to make an optimal decision based on some predetermined policies. Decisions are made using a combination of previously learned information and new information input to the system. A policy will help the system decide which configuration is the best and most optimal to use. The system can also learn which decisions have helped create the best configuration and will be able to use these results in future decision making.

To learn context information, devices will be able to use monitoring and discovery mechanisms that will be able to sense changes in the environment. Monitoring can provide information such as traffic requirements and QoS levels. Context information that is learned can be used to address potential problems and trigger appropriate optimization techniques.
3.2.3 Machine learning to tackle new problems

Application identification plays an important role in modern network management systems. Systems commonly check the transport layer source and destination port to determine the appropriate application. However, as applications can now use various ports, a machine learning approach has been discussed that can create a tree-like model to visualize the application (Wang, 2011).

As thousands of applications now use the Internet as a means of data transfer, chatting, video calls, etc., this means there is a wide variety of protocols being used. Knowing what application protocols are being sent across a network allows for smarter configuration of these networks. Currently, port numbers and deep packet inspection are used to decide what protocol is being used, but with an increased use of proxies and common port numbers being used to bypass firewalls, this is getting harder to do. Another way of identifying these protocols is needed, and a machine learning approach that uses the sequence of packet’s payload length in a network flow has been used in this paper.

The paper suggests that data payload lengths often form unique patterns among applications. With this in mind, machine learning should be able to use this information to classify protocols and determine which is being used on the network. Their results show that machine learning can be used to help accurately determine which protocols are being used. The implementation isn’t included in the paper, but this still gives a clear indication of the problem-solving capabilities of machine learning.

3.2.4 Machine learning applied to oil spill detection

Companies that produce software to detect oil spills in radar images currently use machine learning techniques to help pick out a spill in an image. In this
When the paper was written there were still people that were trained to manually look through radar images and pick out oil spills. This was made difficult because of natural features of the sea in the images such as high winds affecting the radar or algae in the water. There were systems in development, at the time of the paper, which would use previous information to build up a probability of a mark on the image being an oil spill by also using information on wind strength and direction at the time. The authors of the paper want to create a system that can be customized by training on examples of spills and non-spills provided by a user. The user will also be able to control the trade-off between false positives and false negatives.

Input to the system was a variety of images that were processed to find dark patches. A human expert would then specify which the spills are and which the lookalike spills are. The output from this will be a classifier that the software can use to decide whether a dark region is an oil spill or not. The software will use this classifier to prioritize which images needed to be checked by a human to help reduce the human’s workload. However, the total number images presented for inspection can’t be too small so actual slicks aren’t missed. Users should be able to control the number of images displayed to them after the analysis process.

The three problems the authors encountered in collecting training information was the limited availability of radar images of oil spills, unbalanced classes of data and data coming in batches. Unbalanced information can make it harder to find an important event. This happened because most of the images supplied were lookalike spills which meant the system was able to learn a lot about these types of images but not so much about real spills. The paper says 96% of the data was comprised of lookalike spills. The problem with data
coming in batches is that the information in one batch could all be related, so when the next batch is looked at the data could be from a different source or the collecting equipment could be set to a different configuration.

The paper talks about the issue related to formulating the problem. An example of this could be granularity of the system. Would a user be shown the entire image, a suspicious dark region or simply a pixel that could be part of a spill? Another issue was how to present this information and should results be grouped into categories such as grouping lookalikes into subcategories. Another issue is how to decide to between true and false positives so that the results can be optimized in that the number of potential spills shown to the user can contain as many actual spills as possible.

After the problem has been formulated, the next step is to decide on the learning algorithm used. They needed to be careful because of the large imbalance of lookalikes and spills and the fact that the total number of spills and lookalikes in various images they acquired varied greatly. At the time the paper was written there was a number of learning networks being developed but the paper decided on two approaches that were deemed helpful. One attempts to balance the classes by removing any redundant or noisy example in the training data, the other attempts to find an algorithm that is intrinsically insensitive to the class distribution. As each new image was analyzed and each learning algorithm was developed, a new dataset was created. Bookkeeping was in place to keep track of the versions of the datasets.

The paper shows that, when using machine learning, it is important to define what the problem is and what information is to be learned. The paper highlighted that were various learning algorithms in development at the time. One key feature is the mention of multiple datasets being kept. The paper
does not discuss how these are used but one common sense approach would be to choose a dataset that gives the most reliable output.

### 3.2.5 Unsupervised Learning

Sometimes a system is created that is intended to learn without any human interaction confirming if training decisions are correct or not (Alpaydin, 2010). In these cases, all the system is given is some input data and is then required to find regularities in the data. This is called density estimation.

One method for density estimation is clustering, where similar data is grouped together. For example, a store could determine different types of customers depending on what they purchase and then can give discounts or coupons relevant to their purchases. Another example of this is being able to look at newspaper articles and group articles together that contain the same types of words relating to a certain field. For example, sports articles could be grouped together by matching articles containing the most references to a tournament or a sports team.

### 3.2.6 Reinforcement Learning

Reinforcement Learning is a slightly different method of allowing a machine to learn (Furnkranz, 2001). These machines will look at several possible actions and then make a decision on which action will be the most beneficial to be take. Like unsupervised learning, the machine won’t have a supervisor helping with the training data. It will perform the possible actions and then reward the actions that led to the right result. This process is used in a tool called MENACE, the Matchbox Educable Noughts and Crosses Engine (Michie 1961. 1963). This tool was created to learn how to play tic-tac-toe. All 287 possible states of the game were assigned a weight. The machine could then start anywhere and its following moves would be made by using
the weight of each subsequent state. If subsequent states have the same probability then one would be chosen at random. If the game is won, all moves that are made on the way to winning are rewarded by increasing their weight so they are more likely to be chosen again in the next game if the game state appears again. If the game is lost, then the weight is reduced so that these states are less likely to be chosen again.

3.3 Evolutionary Programming

Evolutionary programming forms a branch of artificial intelligence that aims to find an optimal solution to a problem from a finite set of potential solutions using natural selection (Yao, 1999). Natural selection is mimicked by these algorithms in the following ways (Pal, 1996):

- Evolution operates on a chromosome level and not at the level of an entire being. As chromosomes are the building blocks of a being, as is the counterpart in evolutionary programming. These algorithms are performed on possible smaller solutions to eventually build a full final solution.

- Natural selection obeys Darwin’s ‘survival of the fittest’ paradigm where chromosomes with the highest fitness value will be chosen to proceed to the next evolution. In evolutionary programming the smaller solutions that are deemed most appropriate to solving the final solution will be chosen more frequently than those that get further away from a final solution.

- Evolution doesn’t look back at previous evolutions of chromosomes so nature just makes the decision on which chromosomes are the fittest by looking at the current selection it has to choose from. Genetic algorithms also do this; they have an evaluation function that will evaluate each of the current solutions it is presented with.
- Evolution also allows the mutation of chromosomes to help change the current chromosomes in the hope of improving them to create better ones for the next evolution. Genetic algorithms also have this property and can mutate the partial solutions.

To summarize, on a machine, a population of trial solutions is chosen at random, these solutions are then replicated into a new population where the mutations are assessed by their fitness.

Yao gives a guide on how Classical Evolutionary Programming works by splitting it up into a number of steps:

1. Generate initial set of solutions, x
2. Give each solution a rating on how good it is
3. Make each solution generate an offspring, giving us a new set of solutions, y
4. Calculate a rating for each of the solutions in set y
5. Conduct a pairwise union of x and y. Choose a number of opponents at random. If score is better than everyone else, it ‘wins’.
6. Select a number of solutions with the most wins to go into the next round
7. Stop if an optimal solution has been reached. Otherwise, go to step 3

Pal gives these steps in a similar manner in the following diagram:
Yao also talks about ‘Fast Evolutionary Programming’. This is almost the same as classical EP but used Cauchy mutations. These mutations are created in such a way that the new solution they give will be further away from the solution of their parent. This helps ensure a wider solution area and, his study shows, give a better probability of finding the most optimal solution quicker.

This idea could be adapted to help this project learn about changes in the environment. The teacher could choose a configuration that has been used before and then Class Control could take this as a potential model of the current environment. It will then check the nodes in this model to see which of them still exist and discard any nodes that don’t respond. This will then leave the teacher with the configuration of their choice with the most up to date information in it. This configuration will then be stored at the end of the software’s runtime for it to be used again in a future run.

However, the previous paragraph is closer to a machine learning application, than an evolutionary programming one. As evolutionary programming concerns itself mainly with combinatorial problems that solve problems such as the optimal way to deliver...
packages or which taxis should pick up which fares it was deemed not worth looking into more detail about how these algorithms work.

3.4 Network Discovery

Network or service discovery protocols allow a device to detect other devices and services on a network. This requires a common language or protocol that all devices will use to be able to communicate with each other successfully. Networking is the core functionality of this project as it centers on machines being able to communicate with each other. *Class Control* was originally set up for one classroom where the machine names wouldn’t change and could be hardcoded into the program. This project will expand *Class Control* so that it can dynamically discover a new environment while adapting to any changes in the environment. There are three main protocols that this section of the literature review will investigate, the service location protocol the simple service discovery protocol and a technique called zero-configuration networking (Johns, 2002).

- **Service Location Protocol** – This protocol allows computers to find services on a local area network and works on any size of network from small to large enterprise networks. Each service will have an URL that is used to locate the service and it may also have a list of attributes to allow a machine to find out more information about the service by using various queries. This allows machines to find devices of a certain type or that perform a specific function. SLP gives devices three different roles but a device can choose to take more than one of these roles. User agents search for devices; services agents announce services and directory agents cache services.

- **Simple Service Discovery Protocol** – This is almost similar to the service location protocol as it allows the advertisement and discovery of network services and presence information. It does this without
using server-based tools such as the Dynamic Host Configuration Protocol or the Domain Name System. This is most intended for Universal Plug and Play devices and is intended for small networks such as homes or small offices. Multicast addressing is used to allow services to announce themselves.

- **Zero-Configuration Networking** – This is a set of techniques that can automatically create an Internet Protocol network without manual intervention or special configuration servers. This is built on three main technologies:
  
  - Assignment of numeric network addresses
  - Automatic resolution and distribution of hostnames
  - Automatic location of network services

  A downside to zero-configuration is that it uses a multicast DNS which is vulnerable to spoof attacks within the multicast IP address.

### 3.4.1 Project Octopus

Project Octopus is a project developed by Cornell University (Keshav, 2005) to create a tool that can discover the topology of a given network. They have developed this tool to help solve the problem of networks that are growing at an increasing rate. This growth causes problems when trying to troubleshoot errors and add new hardware. The way they go about doing this is in three steps:

**Domain Topology Discovery**: To discover the machines in a given domain, PINGs can be used to discover what machines exist in a subnet. If a machine responds, it is alive and can be added to the domain’s topology. They can also use the Simple Network Management Protocol which allows someone to query a machine or router about its view of the network. The paper notes, however, this
protocol may be restricted in most networks that support it. Another tool used is traceroute which allows one to see what routers a packet goes through on its journey to another machine. A combination of these tools allowed the research team to develop an algorithm to determine the topology of a given network.

**Backbone Topology Discovery**: This is the engine that performs traceroutes and collects the information it finds from the various nodes it wants to find information about and sends its results to the backbone master. The master is then able to ask the nodes for more information, if needed.

**Network Statistics Discovery**: The group uses pings to try and determine extra information such as delay and bandwidth measurements. At the time of the project, the group was still trying to automate this process and discover better methods of acquiring this information.

### 3.4.2 ENTRAPID Protocol Development Environment

Another project from this school is project ENTRAPID (Huang, 1999). The goal of this project is to create a tool that allows a protocol to be tested for robustness by simulating and visualizing the protocol before it is deployed. Protocols can be used on large networks and can be difficult to test if a developer has to create a new OS that adopts the new protocol and then put this on multiple machines to test every new feature or fix each bug. This process could become a painstaking process. ENTRAPID allows a software based development environment creating a simulated test network. The paper talks about how their tools must simulate the network, or the Internet, as close to real life as possible so that any code can be moved straight from the development environment into live use.
While a small step away from network discovery, protocols are still a key feature of *Class Control* so this project still merits a mention here.

### 3.5 Existing Software

In the development of *Class Control* some existing software was looked at to gain some inspiration for functionality to add to the software. This section will go back to these existing pieces of software to see how they are installed in a classroom. These are designed to be installed in any lab, rather than being hardcoded to work in just one so learning about the installation process could help with this project. There are a number of existing software and technologies that could be used for the core part of *Class Control* involving the screen updates and remote control. The first paper for this project (Grimes, 2011) talks more about existing software and technologies. As *Class Control* was originally built with the following pieces of software as a starting point, this section will not necessarily look too much at how they work but will look at how they get installed in new labs.

#### 3.5.1 AB Tutor Control

AB Tutor Control is a piece of software that allows a teacher to view and manage a classroom of machines using the Windows operating system. *Class Control* originally got its inspiration from AB Tutor as this was the software being used in one of the labs at the University of St Andrews. Instead of looking at what the software does, this section of the literature review will look at the AB Tutor’s manual to see how the software gets installed.

AB Tutor Control gives a list of pre-installation checks that need to be satisfied in order to successfully install the software. These checks cover ensuring there is an internet connection to validate the software and ensuring the installer has administrative rights on the target machines. The requirements also specify that ports 5151 and 5152 have to be configured to
allow communication through these ports if there are any firewalls or security software in place that might prevent this. A working network connection is also needed between the teacher’s machine and each of the client’s machines. The manual also suggest that there is a possibility a user may want the tutor software and the client software on the same machine. The user is able to do this but won’t be able to run both at the same time. Also, if the teacher software is on a machine that students have access to then care must be taken to ensure they don’t have easy access to the software.

AB Tutor has the ability to create policies for determining which members of staff will be able to control which classroom. Different levels of functionality can also be given to staff, for example removing the ability to change the software’s settings.

![AB Tutor Installation](image)

Figure 9: AB Tutor Installation

The client software is installed on each machine individually and a machine in each room is designated to be the teacher’s machine. Each teacher will see their own class but the administrator will be able to see all of the machines in each classroom.
3.5.2 Apple Remote Desktop

Apple Remote Desktop is a similar tool to AB Tutor but it is designed for the Mac operating system. The client software already comes installed on most versions of the OS so the installation guide is different as only the tutor’s copy of the software needs to be installed and activated. As it is already installed, port numbers and access through a firewall have been configured by the operating system.

The main difference with the installation process is that the ARD manual suggests the ability of using command line tools like scp to install the software remotely using SSH. It also suggests adding the installer to an image of an operating system so it will appear on client’s machines when they log in.

3.6 Conclusions

After reading several papers that relate to various aspects of this project, ideas can now be extracted from them that can help guide this project and aid in creating a set of requirements that this project must attempt to meet. Ideas gained from this chapter will also no doubt help with the design process of this project.

The adaptability section of this chapter shows that this project will mainly focus on intra-individual and environmental differences. There are differences that will make the software adapt to changes such as the user requirements, i.e. if the teacher only wants to view 4 machines instead of 70 machines, and changes related to the environment, i.e. arrange the layout of the machines in a grid in the GUI if that is how the classroom looks.

The sub-system approach to gaining more information for an adaptability mechanism is a pretty interesting concept that could be used in my project as each classroom could ‘elect’ a machine in the room to collect information about the classroom. This would mean that a teacher could query just one machine to find out information about
the whole room; this is clearly an advantage over asking every machine in the classroom for information. This would also therefore involve some form of election algorithm and a way of collecting information about a classroom.

The MUSIC project is closely related to this thesis project as the intention of this project is for it to run on a wide variety of Java enabled devices. The software will not need to change its level of functionality as it will always provide the same options, however, the software would adapt to the number of users using the system at one time. If a teacher only wants to connect to a small amount of machines, the software would adapt itself to increase the refresh rate of screenshots being sent across the network. Likewise, if the software is being used across a large number of machines, the refresh rate will be reduced to ease the burden on the network. Another application the MUSIC project would provide to this thesis builds on the previous idea of having sub machines doing some work. Machines would need to learn about changes in their environment to make sure a machine in the room is collecting information about everyone else. The machine collecting the information about the class also needs to ensure it updates itself if a student’s machine is turned on or off.

The questions from the model approach paper can be answered in relation to this project:

1. **Under what conditions does the system undergo adaptation?** The software will need to adapt upon startup to first establish information about the machines in a classroom to connect to and will then need to continue to adapt to changes in the state of the classroom throughout the running time.

2. **Should the system be open-adaptive or closed-adaptive?** An open-adaptive system will be able to adapt and change its behavior during runtime whereas a closed adaptive will make the changes ready for the next runtime. As this project adapts to the current state of the lab and
adapts the GUI wherever it can, it is open-adaptive. It could also be though as closed-adaptive because these changes to the configuration are stored and used next time.

3. **What type of autonomy must be supported?** The system will need to be as automatic as possible with as little human interaction as possible

4. **Under what circumstances is adaptation cost-effective?** The benefits of adaptation will outweigh the performance and memory costs because the adaptation will not only keep the software learning about the environment but will also optimize performance.

5. **How often is adaptation considered?** The adaptation changes will try to be made as often as possible but only testing and evaluation will help to decide on the optimal adaptation frequency.

6. **What kind of information must be collected to make adaptation decisions?** The teacher’s software and the student’s software will both have decisions to make throughout the life of the program. The students will have to ensure there is always one machine in the room collecting information for the teacher. This information will include data such as: which computers are running, the dimensions of screen sizes, port numbers and addresses. The teacher will query this information on a regular basis to make decisions on how to adapt to changes.

As well as addressing how to make the software adaptable to its changing environment, care must also be taken to ensure the software can detect and recover from system failures. Machines could freeze or disconnect from the network. The software must be able to handle faults from other machines and also recover from its own unexpected faults. The software will always try to remain in a safe state at all times. A safe state would be a machine running with no problems, knowing up to date information about all of the other machines in the room.
Security concerns were also raised as an issue so the software will be able to check that messages are using the correct protocol and are coming from a trusted source. If anyone receives invalid messages or if someone tries to impersonate a teacher machine, the software will be able to pick up on this and report back to the person using the software. This is particularly important when it comes to viewing the screen of a machine that isn’t your own. Only authorized users will be able to view and take and control of remote machines.

As previously mentioned, machine learning will enable the software to learn about its environment but this information will need to be stored somewhere in an easy to use format. One method discussed of storing this data was in a tree format. This will be applied to this project as the teacher software will try to discover the sub-system in each classroom. This sub-system will then provide the teacher with information about each machine in the classroom. The teacher will then build up a list of classroom names and individual machines to allow the teacher to choose who they want to connect to. A tree structure will allow each classroom to be a node and each of the machines will be the leaves, XML is a good language for this and will be chosen as the format for the database to be stored in.

The service level protocol fits this project because the teacher machine will want to discover where the student machines are. The teacher may also want to only look at machines in a certain classroom or even narrow their selection further down by operating system or usernames of people logged in. The teacher would act as user agent while searching for machines and also a directory agent to store information and learn from it. A student machine would act as a service agent to announce its availability and attributes. A sub-system architecture would require that a student machine in each classroom also acts as a directory agent to store information about the other machines in the classroom.
Simple service discovery is useful for Class Control because it doesn’t need to use tools such as DHCP or DNS to set up its knowledge base about the location of machines. Student machines can use multicasting to allow each sub-system to build up information about the machines in their classroom. Each sub-system can use multicasting to announce their location to the teacher, or each student could use multicasting to point the teacher in the right direction to each sub-system. A teacher could appear anywhere in the network and multicast will allow the teacher to be discovered as fast as possible.

Zero-configuration could be incorporated for Class Control as self-adaptability and self-learning would require a mechanism that allows changes in the network to be discovered without any manual checking or input from a user.
4. Requirements Specification

The objectives section combined with the literature review have helped establish a list of requirements that this project must meet. This has been split into two sections. The first will give a brief overview of the various components of the system. From these, section 4.2 shows a structured list of requirements that has been created.

4.1 Functional Overview

*Class Control* has been developed in a way that has allowed it to consist of various components, which were developed independently and then combined to form a single system. The modularity in this design decision gives the ability to add new components and modify or remove existing ones.

The system has been split into two main programs:

1. The main teacher program (server) containing most of the implementation
2. The student (client) software that exchanges commands with the teacher

This thesis project will update the functionality of both parts of this software so this report is still able to split them up into two separate programs working together. As well as updating the software, the protocol that allows messages and commands to be exchanged will also have to be updated.

4.1.1 Teacher (Server)

The core components that will be added or updated in the server application and the design process of their architecture are described below. Various diagrams have also been added to help explain these features in greater detail.

4.1.1.1 Finding Students (Clients)

The figure below gives a basic object diagram of the teacher application listening for clients. A component will listen for messages from clients that contain information about the client and this information will be stored in the XML database.
When the server software is first started the teacher can choose to either spend a period of time waiting to be contacted by clients, or skip this and use a previous configuration. When the initial scan is skipped, there is a scan that runs for the whole time *Class Control* is running. In either scan, once the server receives a message containing the client’s IP address and classroom name etc. it will add this information to the XML database and begin building up information about what classrooms and machines in them have been discovered. Once this phase is complete the teacher will be presented with a screen to make a decision on which classroom they want to connect to. The figure below helps visualize this process:
4.1.1.2 Gathering information about a classroom

The server will have an XML database that will be storing information that is received from the clients that communicate with it. The first phase of the software loading will just collect information so that the teacher can be presented with a list of classrooms and the number of machines it has found in the classroom. The figure below helps visualize the software loading and starting a thread that will scan for client information.
The software will not just present new information; it can also use information from previous runs of the software, if the teacher wishes to choose a previous configuration. The database collects information throughout its runtime from not just the classroom it is connected to but also to clients in other classrooms that are looking for a server application. Every time a message from any client is received, the database will be checked to see if this is new information. This will help prevent stale or duplicate information in the database.

When the teacher first loads the software, they will also have an option to load past configurations. If a teacher knows of a past configuration that worked well or they know already has information about a whole classroom, the teacher can choose to save time in waiting for the server application to gather information about the class before connecting to it.

This means that multiple configurations will be stored containing snapshots of the system at each time the system was closed. Checks can be put in place to ensure only a certain number of past configurations are stored and this will ensure only the most recent and most recently used configurations will be available without taking up too much space.

There are three main stages of how the server will gather information about its environment:

1. The server will start its initial scan on startup looking for the existence of any clients around the school
2. The server may skip the initial step and go straight into loading a previous known good configuration.
3. Throughout the running of the software, the server application will look for new clients appearing across the network and will be able to add new clients to the GUI if they are in the classroom the teacher is working with
All information collected is stored in the database so that it can be used during the running of the software and also in future use of the software.

**4.1.1.3 Letting the user choose a classroom to view**

A new part of this project will allow the teacher to have the ability to choose from a number of classrooms to connect to. This will be presented to the teacher as part of the GUI and will allow the teacher to view a list of the names of classrooms that have been found and the number of machines in them. Using information that may have already been acquired, the teacher will be shown the number of machines currently found in the room.

When a teacher chooses a classroom to connect to, the software will begin contacting each of the clients. A connection can then be made to begin the sending of screenshots to the server application. If machines from previous configurations aren’t available, this will be updated in the database accordingly. The figure below shows these steps in choosing a classroom, notifying clients and updating the database. It shows how heavily involved the database is in the running of *Class Control*. 
4.1.1.5 Fault Tolerance

When communicating with machines on a network it is likely that problems such as connectivity or unexpected system failure could occur. *Class Control* will be able to monitor problems like this by checking if the machines it is connected to are still there. There are a number of checks the software will be able to perform. When in thumbnail mode it can check to see if a screenshot is being received regularly. When about to take control of a machine or send a message to it, if the message doesn’t go through, the server will be prevented from making any more contact with this machine until it contacts the server again. If the machine loses connectivity during a remote control session, the remote control window will be closed. If a machine is discovered to have been lost, the GUI will be updated to reflect this.
It is also possible that teacher’s machine could have a problem. In this case the software will present the teacher with a message saying connectivity has been lost. The program won’t start if it can’t access the network but the teacher will be notified of this. Clients will also have their own fault checking functionality to make sure the server application is running when it thinks it is connected to one. This will be in the form of a pinging mechanism so the client can regularly check to see if the teacher application is still there.

4.1.1.6 Dynamic Screen Refresh Rate

When classrooms get larger in size, the server will be receiving more and more information from the each of the clients as they will constantly be sending a screenshot of their screen to the server. This will not only increase the workload of the server but will also increase the amount of traffic on the
network. To combat this, *Class Control* will be able change how fast the clients send their screen updates to the server. For every 10 machines connected to the network, a second will be added to the refresh time. This can change as clients join or leave the room so that the server will be receiving updates as fast as possible while trying to prevent the network getting too busy.

### 4.1.2 Student (Client)

The client is also its own piece of software. It doesn’t have to do as much as the server but the client still has to carry out an important task to ensure communication with the server.

#### 4.1.2.1 Finding the teacher

The client will continuously send out its information to the multicast address and will wait for a response from the server to begin the connection to a teacher’s application. When the teacher is finished, the client will revert back to its initial state.

#### 4.1.2.2 Establishing a connection

When the server responds, the client begins getting ready to make a connection with the server by opening some sockets to begin sending screenshots to the teacher. This process is explained in the design chapter.
4.1.2.3 Check if a server application opens on the same machine

The client software will potentially be installed on every machine in a classroom which means a teacher may want to use one of these machines. The server software wouldn’t be able to run properly if the ports it needs are already in use by the client software. To combat this, the client and the server will need a check to see if the server software is trying to be loaded. If it is, the client will ‘hibernate’ until the server is finished so it can resume looking for another instance of the server application to be run. This check wouldn’t be needed if the teacher’s machine was always designated to one machine in the room but this may not be guaranteed.

4.2 Structured Requirement Specifications

Section 4.1 allows for the creation of a structured set of requirement specifications. The requirements are linked to their objectives and come with an importance rating of low to high. Low means the requirement is part of a tertiary objective and its completion isn’t of high importance for the scope of this project. Medium and High requirements should be completed with High given the most priority. These requirements will be used to evaluate the success of this project.

<table>
<thead>
<tr>
<th>Requirement #: 1</th>
<th>Requirement Type: Functional/System</th>
<th>History: Introduced 09/11/12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: Self-Discovery of classrooms and machines in them</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: To meet the self-configuration and machine learning objectives, the server must be able to discover other machines on the network to build up a database of machines the server can connect to. This will be done using a multicast address, to be agreed upon. Clients can send a multicast message so that a teacher machine can discover them.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Project Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Importance: High</td>
<td></td>
</tr>
<tr>
<td>Requirement #: 2</td>
<td>Requirement Type: Functional/ System</td>
<td>History: Introduced 09/11/12</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Description: Clients must make themselves known to the server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: To aid with the self-configuration, machine learning and self-adaptability objectives the server must be able to discover clients. The clients must make themselves known to the server. This needs to be done so that the teacher machine can collect information about the clients.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Project Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Importance: High</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: 3</th>
<th>Requirement Type: User</th>
<th>History: 09/11/12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: User is to be presented with a GUI which will display the layout of the environment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: For the teacher to be able to choose which machines to connect to and to verify machines have been discovered, they will have to be presented with a GUI displaying this information. This will also help a teacher load a previous configuration that will satisfy the loading of previous configurations objective</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Project Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies: 1,2</td>
<td>Importance: High</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: 4</th>
<th>Requirement Type: User/ Functional</th>
<th>History: Introduced 08/14/12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: Keep full existing functionality of Class Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: As this project is aimed to extend the functionality of Class Control and not reduce it, this requirement must be met. Class Control is already able to broadcast a screen to a classroom and receive screenshots from each of the clients and is a fully working piece of software.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Self</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Importance: High</td>
<td></td>
</tr>
<tr>
<td>Requirement #: 5</td>
<td>Requirement Type: Domain</td>
<td>History: 08/14/12</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Description: The system will not store information about students</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: As with the original Class Control, any information the server receives about students from client software should not be saved as students have the right to a level of privacy in the classroom. Only the teacher has the authorization to see what a student is doing during their class. Any information the student inputs to the machine they are using will not be stored in any way as this could contain personal information. Meeting this requirement will satisfy the security objective.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Ethics Domain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Importance: Medium</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: 6</th>
<th>Requirement Type: Functionality/System</th>
<th>History: 09/11/12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: The system will still need to be scalable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: The environment could potentially consist of the whole university so ideally the software would be able to cross subnets to discover more machines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Self</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies: 1, 2</td>
<td>Importance: Low</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: 7</th>
<th>Requirement Type: Functional</th>
<th>History: Introduced 09/11/12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: The system will be able to run on any OS that can run Java</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: Class Control was originally designed for the Mac OS X, but this project will enable this project to run on any OS that allows Java programs to run</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Project Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Importance: High</td>
<td></td>
</tr>
<tr>
<td>Requirement #: 8</td>
<td>Requirement Type: Functional/ System</td>
<td>History: 09/11/12</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>Description:</strong> The system will be able to handle unexpected errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Rationale:</strong> Part of the adaptability objective of this software is to handle errors that could happen such as network issues or a client's machine crashing for example. A list of errors this project will be able to handle will be defined in the implementation section of this report.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Source:</strong> Project Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dependencies:</strong></td>
<td><strong>Importance:</strong> High</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: 9</th>
<th>Requirement Type: Functional/ System</th>
<th>History: 09/11/12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> Dynamic Screen Refresh Rate</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Rationale:</strong> Another part of the adaptability of this software is to change how fast screen shots are refreshed to ease the burden on the network in large classroom sizes, if needed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Source:</strong> Project Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dependencies:</strong></td>
<td><strong>Importance:</strong> Medium</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: 10</th>
<th>Requirement Type: System</th>
<th>History: 10/25/12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> Sub-systems will collect information about the classroom they are in</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Rationale:</strong> Instead of the server collecting information from every machine in the network, sub-systems could collect information about their classroom and then pass this to the server. This is listed as a tertiary objective and does not have to be implemented.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Source:</strong> Self</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dependencies:</strong></td>
<td><strong>Importance:</strong> Low</td>
<td></td>
</tr>
<tr>
<td>Requirement #: 11</td>
<td>Requirement Type: Functional/User</td>
<td>History: 10/25/12</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Description: The system will keep an audit trail</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: Mentioned as a tertiary objective, an audit trail could be kept to let system administrators know what machines are being connected to and which machines are connecting to them</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Self</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: 12</th>
<th>Requirement Type: Functional/ Users</th>
<th>History: 09/20/12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: Server must store previous configuration states for use in future use of the system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: The system can save previous states as it learns about its environment each time it is in use. The teacher can be presented with previous configurations to help speed up the process of connecting to a classroom. This is required to meet the secondary objective, loading previous states.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Project Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies: 14</td>
<td>Importance: Medium</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: 13</th>
<th>Requirement Type: Functional/ System</th>
<th>History: 09/11/12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: Server must be able to handle clients joining and leaving the environment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: Clients could add or remove themselves to the classroom in use and the teacher will want to see these changes reflected in the GUI. This is to help with the adaptability objective of the project</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Project Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Importance: High</td>
<td></td>
</tr>
<tr>
<td>Requirement #: 14</td>
<td>Requirement Type: Functional/ System</td>
<td>History: 09/11/12</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Description: Information about classrooms and machines in them will be stored in XML format</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: Information collected needs to be stored in XML so it is easy to access and modify and is also human readable. This will help with loading previous configurations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Project Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Importance: High</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: 15</th>
<th>Requirement Type: Functional/ System</th>
<th>History: 09/11/12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: Existing protocols of Class Control must be adapted to ensure new information needed by the server about clients is interpreted correctly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rationale: The original Class Control has a protocol in place for sending messages but the teacher will need more information about the client such as the classroom name, its position in the room and the machines screen size. This needs to be met so all of the primary objectives can be accomplished</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source: Project Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Importance: High</td>
<td></td>
</tr>
</tbody>
</table>
5. Software Engineering Process

This chapter describes the software engineering methodologies and techniques that have been used to ensure that Class Control was not only completed on time but also contained the required functionality. This section also contains a discussion and how planning for this report and keeping the code to a set standard were carried out as these are also vital processes to ensure this project’s successful completion.

5.1 Requirements Engineering

Choosing the requirements was carried out with the project supervisor after the literature review had been completed. The literature review provided an insight on various methods for solving the original problem of this thesis and was used to help create the requirements in chapter 4. The requirements in this document are described in natural language terms with a rationale as to why these requirements were chosen. This has been to be as clear as possible what the requirements are and why they are required for this project’s success. To help with the design process and define a priority ordering, the requirements are also given a rating of high (red), medium (orange), low (yellow). This priority ordering also helps the evaluation process to see if the most important requirements have been implemented. Low priority requirements aren’t vital to the project’s success but would improve the project’s overall functionality.

5.2 Methodologies Used

There are various methodologies used in software engineering to guide the process of creating a software system. However, because of working alone, not all of these are appropriate for the development of this system. A combination of two methodologies to create Class Control was chosen, these were Scrum and prototyping. Using these methodologies allowed Class Control to be built on time while also considering the robustness and reliability of each component along the way.
5.2.1 Prototyping

A prototype is any attempt to realize any aspect of software content [Arnowitz, 2007]. Prototypes allow for components of a piece of software to be created and evaluated on their usability for a final project. This also allows developers to create sub-systems to show clients potential solutions and to experiment quickly with various methods to create a better final system. A key point for a successful prototype is that there isn’t too much attention to detail and features. Prototypes are generally created fairly quickly to provide a proof of concept and then discarded and rebuilt to create a more robust final system. The thought process of prototyping can be seen in the diagram below:

![Prototyping Methodology Diagram](image)

Class Control is built as a combination of various components that work together to provide the full functionality of the final project. This modularity allows for different components to be developed separately and then added to the final system. This means prototyping is a good technique for this project to test different methods to complete the various tasks that need to be carried out. For example, a small multicasting client could be created that can send out messages, and another small piece of code that will receive the messages and parse them. Once this piece of code meets the required level of functionality, this knowledge can be used to then develop this functionality.
into the final project and the next step, which would be adding the parsed message to the database, could be worked on. These small parts of functionality, once shown to work could then be discarded and a more iterative development technique could be used to add them to *Class Control*.

### 5.2.2 Iterative Development

While some components can be tested using prototypes there still has be a stage where these components are integrated into the final code base. A more iterative approach allows a component to be added to the software while still keeping a fully functional piece of software. Some of the components may require some modification to other components in order to cooperate properly with the rest of the software. Iterative approaches to development allow for small updates to be made and tested ensuring each step of the development provides fully functioning code.

There are a number of iterative approaches but the approach that seems to work best for this project is called Scrum (Lacey, 2012). Lacey says it is ‘a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value’. This statement shows one of the key reasons Scrum is useful in this project: it allows requirements to be changed throughout the development phase. The supervisor has set primary requirements at the start of the project but as it develops the method in which these requirements are implemented could change depending on how other components are implemented.
Figure 18: Scrum Framework

Figure 18 gives a visual representation of the scrum framework. There are usually three roles in a Scrum framework: the Scrum Master, Product Owner and the Development Team. In this project, the supervisor will act as the product owner and the scrum master. The supervisor has set the requirements but in weekly meetings will act as the scrum master guiding the author of this project on what problem to tackle next or to provide help with previously set tasks. The scrum framework consists of sprints where the development team, the author, will carry out the tasks set by the scrum master. The figure shows a 30 day sprint with a meeting every 24 hours. Because the size of the development team is one person and the size of the project, a weekly sprint with a weekly meeting has been enough to complete the project but the ability to have extra meetings was there if required.

5.3 Timing

Figure 19 shows a simplified Gantt chart illustrating the expected amount of time spent on each section of the project. Timings slightly adjusted throughout the year either when encountering problems or when other work had to be completed. Because of this, these timings weren’t strict deadlines but using them as a guide helped ensure the project was completed on time. The colors
on the chart relate to the various sections of the project shown in the key in figure 20:

<table>
<thead>
<tr>
<th>ug</th>
<th>Sept</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
</tr>
</thead>
</table>

Figure 19: Gantt Work Flow Chart


Figure 20: Work Flow Chart Key

5.4 Weekly Meetings

Each week, a weekly meeting was held with the project supervisor to discuss each step of the process. Meetings would include discussions on what work had been done during the week, what was proposed to be done by the following week and what changes needed to be made to previously completed work or code. This helped keep the project moving at all times and allowed the supervisor to be constantly kept up to date on where the project was and where it was heading.

5.5 Coding Standards

The code was split up into packages and classes to keep it easy to maintain and navigate. Classes are given relevant names to their purpose and methods inside them are well commented to allow anyone else looking at the code to quickly understand what the code is doing. This standard of breaking up the code and commenting everything was a standard set in the first project and has been continued in this project to ensure future work can still continue.
5.6 Mercurial Repository

As this project could become very large and complicated it was decided to use a version control system. CVS was the original method of having a version control system, but after the first stage of this project was completed, St Andrews moved the project into a Mercurial repository. This is a slightly different version control system but the reasons for using one are still the same. A version control system allowed the ability to make additions to the code and if they didn’t work, it was possible to roll back the system to a previous version. Mercurial also allows the code to be downloaded on any machine so that it could be worked on at home or in a lab. When a new feature had been implemented and tested, it could then be committed to the server.

5.7 Testing

Planning how to test the system was also an important part of the design process. Testing was important for this system because in the environment that it would be running, the teacher would not be able to see if the software is running correctly on the students’ machines as their screens are facing away from the teacher. This means that the teacher needs to be confident that the system is working as intended and will be outputting the expected behavior.

It was decided a context-driven approach would be appropriate in testing the operation of the system. This is a paradigm of developing that involves debugging software by looking at how the software is going to be used in the real world. This was because nearly all of the events that take place in this system can be seen in the GUI that has been created and this is all a teacher would see. This isn’t just a ‘if one can see it working then it must be working’ approach because the GUI was only the first point of checking to see if the system was working as expected. For every component, there was a series of messages that would be output to the console
window in Eclipse. This shows that the code executes how it is intended, show when threads start and end and what messages that are being sent and received contain.
6. Ethics

This section takes a look at the ethical considerations made in the development of Class Control. As this is a tool that can show a teacher a remote view of a student’s machine, care has to be taken to ensure this student’s privacy isn’t invaded (Ermann, 1997).

There are some potential risks with having Class Control installed in a lab with multiple machines. These could be:

- A teacher could peek at a student’s instant message conversation with another person
- A teacher could look at a student’s screen without them knowing and without telling them
- There is the ability for a student’s keystrokes to be sent to the teacher’s machine so the teacher could obtain passwords and sensitive information
- The teacher could take remote access of a student’s machine and access a website that may have already been signed in to by the user
- An unauthorized user could take control of the teacher’s application without the student’s knowing and perform some or all of the above actions

These four points clearly highlight some of the implications that come with the development of Class Control. The following policies are intended to prevent these problems:

- As students are supposed to be studying in their class time, when Class Control is to be used, the teacher is to be instructed to tell the students in the classroom the software is being used. This will let students know that the teacher is monitoring them and could see their conversations.

- A student will always know if their client is running because an icon will be displayed on the student’s toolbar showing this. This also helps with the first bullet point because the student will know if there is a chance of them being monitored or not. In the development of Class Control, care was taken to ensure no computers were ever accessed that were being used by anyone
other than the author. This was easy to enforce because the teacher application would only receive screen shots from machines running the client software and it was never installed across the lab on every machine during development.

- The teacher would not be able to access a website that the student has signed in to or make illegitimate Facebook updates, for example, because the student would be able to see what the teacher does when they take remote control and the student can also still control the computer while the teacher has control. The student can also lock their screen and lock the teacher out also

- A password has been added to Class Control so that only an authorized teacher with the password can access the software

With these considerations in mind and making sure a student’s privacy is never compromised, any ethical concerns raised with Class Control should be satisfied. Because of the component based nature of Class Control it is very easy to change the functionality of the software to adapt to any ethics requirements that may need to be met in the future
7. Design

Now that the system’s main functionality has been described, the paper will begin to look at how the system is going to be built and what components will be included the system architecture. This chapter will explain in detail how the system architecture will be comprised and then will go into detail about what each component will have to do in order to meet this project’s requirements.

7.1 System Architecture

![System Component Diagram](image)

Figure 21: System Component Diagram

Figure 21, above, shows that there are two main components, one used by the teacher and one run on the student’s machine. These two components interact with each other using the protocol described in section 7.2. This section of the paper will look at each component in more detail.

7.1.1 Teacher

The teacher’s application, the server, provides the most of the functionality to *Class Control* and does a lot more work than the student’s client as it has to interact with multiple clients at the same time. A client would only interact with one server at a time. The various components that help the server carry out its tasks are explained below.
7.1.3 Model – View – Controller

Behind each of these components will be code working behind the scenes ensuring that these components cooperate with each other and perform the expected operation of *Class Control*. The Model – View – Controller architecture, MVC, is a good way of representing *Class Control’s* architecture. The MVC architecture separates a piece of software into three main components.

**Model:** The model represents data or processes such as a database or the logic or state of the system

**View:** Represents a visualization of the current state of the model

**Controller:** Provides the ability to manipulate the state of the system by telling the model what it needs to do

The image below gives a visual representation of this architecture.

![MVC Architecture Diagram](image)

Figure 22: MVC Architecture

*Class Control* could also be thought of as an MVC system. The GUI would take on the role of the view. User actions are received by the controller, and user updates are forwarded to the model to make changes to the system state. The controller tells the model what changes it needs to make to satisfy the request of the user.
7.1.1.1 GUI

*Class Control* depends heavily on a GUI that stays up to date and shows the teacher information about the classroom. The GUI will allow the teacher to choose which classroom it wants to connect to, which machines in said classroom it wants to connect to and also show the teacher what the machines are doing once a connection has been made. *Class Control* already has a GUI from the first phase of this project, but additions will have to be made to allow the teacher to choose between classrooms and various configurations. The GUI will also need to be able to handle clients with different screen sizes and classrooms that aren’t in a grid layout like the lab in St Andrews that this software was originally developed for.

7.1.1.2 Database

As mentioned in the design overview, the server is going to store a database of all the information it learns throughout its run time. The database is going to be stored as an XML file. The schema of which is described in section 7.4. The decision to use an XML file as a database was because of the tree like structure it provides to give a clear view of the classrooms of a building, if multiple classrooms are stored. The tree like structure also allows for quick information retrieval and modification.

The design overview explained that the databases will constantly be updated throughout the lifetime of the server. When the server is quit, the configuration file is saved, allowing the teacher to use the same configuration file the next time the server is run. The server will save multiple configuration files allowing the teacher to use any one of these.
7.1.1.3 Network Commands

There are three categories of network commands used in *Class Control*, TCP, UDP, and multicast. Each of which has their own distinct purpose for the successful operation of the software.

**UDP**: This is used for the sending of screenshot images as it is not a major problem if a screenshot is dropped. This is the same reason VoIP uses UDP.

**TCP**: This protocol is used for the sending of commands from the server to an individual client. For example, to tell the client the server wants to take control. It is important the client receives this message and TCP guarantees this reliability. Section 7.2 will give an overview of the protocols used by the TCP control messages.

**Multicast**: The server will have a multicast socket listening for messages from any of the clients that are multicasting their presence to the environment. Multicast is used because the location of the teacher is not guaranteed so the clients will use a fixed multicast address that the server can use from anywhere it is run.

7.1.1.4 Configuration File

For the network communications to take place, the port numbers and multicast address need to be stored in a file that can be edited without having to recompile the whole program. The configuration file allows this. Currently it is a text file that is read by the server to determine which ports to open or listen on. A GUI could be created to allow edits to be made that won’t leave the configuration file in a state that can’t be read. It may be possible that a teacher or administrator may not know how the configuration file needs to be laid out; an incorrect configuration would affect the running of *Class Control*. 
7.1.1.5 Audit Trail

An addition to this project will be the new audit log feature. Section 7.5 of the report will look into more detail about the audit log and the justification for why it is being added to this project. It will be stored as a text file that can be easily read and understood by an auditor.

7.1.2 Student

The student’s application (the client) doesn’t have as much functionality as the teacher’s application but it still plays a critical role in the operation of Class Control. The client has to search for a teacher application being run and communicate with the teacher by sending screenshots or handling remote control commands. The only part of the student’s GUI is a chat box and an icon to show the student application is running. The figure below shows the current state diagram of the client software, which starts in a ‘Waiting for teacher’ mode and ends there when a teacher is finished with the client. This state will be important in adding the functionality needed to communicate with the new version of Class Control.
The ‘waiting for teacher’ state will be the only state that needs an extension. The rest of the client’s functionality is covered by the other states. The waiting for teacher state will now be the state that does the broadcasting to look for the teacher. The teacher will still send its initial ‘SayHello’ and final ‘TeacherBye’ messages to connect and disconnect from the client but the client will then start and stop its availability multicast accordingly.

7.1.2.1 Network Commands

Like the teacher’s application, the student application deals with three different transport protocols for sending and receiving messages. The protocols for the messages that will be sent are shown in section 7.2.

7.1.2.2 Configuration File

The student application will also have a configuration file saying what ports and multicast address to use. This will be the same as the teacher’s.
7.1.2.3 Audit Trail

An audit trail will also be kept on each of the student machines. This will keep a log of which machines connected to the client, if they took control and any chat messages that were exchanged between the teacher and student on this machine. The audit trail is explained in more detail in section 7.6.

7.2 Network Protocols

*Class Control* relies on a series of protocols that allow information to be sent and received between the server and the clients. These protocols specify what each message means and what action to take upon its arrival. The protocol already in place is described in this section. Below is a diagram showing what classes and threads are involved in the communication part of *Class Control*. Thread management is also an important part of this software. The figure has the threads colored in to show which threads need managing. The ones in red are the only threads that run the whole time the software is running. This is to ensure a message can always be received by a client or a server at all times.
7.2.1 **Teacher Control Server**

A student can send the teacher one of the following messages. The student’s IP address can be used to uniquely identify the received messages. The message will contain one of these keywords.

**here** - This lets the teacher know that the client is running on a given machine. The teacher’s application then sends a message back asking for the username of the user.

**name** - The name keyword is followed by the username of the person running the client software. This is then added to the client’s information stored by the server so it can be displayed to the teacher.

**chat** - This lets the teacher know that any text after this keyword is a message to be shown in the chat window for this client. If the teacher’s chat status is busy then the teacher will reply with an
automatic busy message. Otherwise, the chat box will be shown to the teacher with the message that has been sent.

**goodbye** - This lets the teacher know that this client’s program has been shut down and is not there anymore.

**PING** - The ping message is different because the teacher application is not interested in this message as it is just for the client to check the teacher is still there so it is just ignored.

For extra functionality that this project requires, an extra message will be added to the protocol that the client will send to the server. This is so the server can build up information about clients in the environment. This will contain information that will be stored in the database so the initial broadcast message will be:

“BC CLASSROOMNAME ROW COL TOTAL ROWPOS COLPOS SCREENX SCREENY”

The teacher will parse this message and store the information in the database.

### 7.2.2 Student Control Server

The student will also receive commands from the teacher so that it knows when to carry out certain tasks. These commands are listed below:

**startTCP** - This command tells the student that the teacher has opened a single viewer window to view this computer’s screen. This makes the client send screen updates at a faster rate and also sets up a new TCP connection that the teacher can use to send remote control commands.

**stopTCP** - When the teacher has closed the single viewer window, this command is received to tell the student to close the TCP connection that was created and to slow down the screen updates to stop congesting the network.
**startMC** - When this message is received it will tell the client that it is to display any future received broadcast in full screen.

**stopMC** – This will tell the client it is exempt from having to show any future broadcast images in full screen mode.

**gobroad** - When the teacher has a single viewer window open it can send this command to make the client broadcast its screen to the rest of the class.

**stopbroad** - This stops the broadcast that the command above started.

The client will also need to handle a new message and this will be the message **streamPORT** where PORT is replaced by a port number that the client will use to send the screen shot updates to as each client needs their own port to do this. When a client receives this they know a teacher is present and can begin the original connection process.

### 7.3 Configuration Files

*Class Control* already comes with configuration files. The image below shows the configuration file of the server as it stood after phase one of *Class Control* was completed.

```
StreamPort=58999
TeacherControlPort=58996
StudentControlPort=58998
Password=SoLbbU372xbU9FEnU4UbeeVAug\=
MulticastPort=58997
MulticastIPAddress = 239.255.0.1
StudentStreamBasePort=59000
```

Figure 25: Properties File

This configuration file allows an administrator to change any of the values without having to recompile the whole program. What each of these mean will be explained in the implementation section in Chapter 8. Both the server and the client have their own configuration file.
7.4 XML Database

The server will maintain a database storing information about every client it encounters throughout its execution. The XML format was chosen for its ease of use in Java, the ability to quickly retrieve information thanks to its tree like structure, and the ability to label nodes with attributes. This tree structure keeps the data organized and readable by humans. The information in the database could be useful during an audit, so being readable may be helpful.

7.4.1 XML Schema

The database will have a schema that it will adhere to so that the server knows the format of the information and where to find it. The basic structure of the schema is shown below. The tags show the nodes of the ‘tree’ and the values within the <> tags show the attributes inside the node.

```xml
<Classrooms NAME>
    <Classroom NAME SIZE A B>
        <Machine X Y IP WIDTH HEIGHT></Machine>
    </Classroom>
</Classrooms>
```

The schema above helps show what information will actually be stored in the database. A school will have multiple classrooms so each classroom will have its own name to identify it. The classroom will also have a number of machines and, if applicable, how many rows and columns of machines there are in the class. This is to help with the layout of the GUI so the layout of the clients on the screen will reflect their position in the classroom.

Each classroom will also have a number of machines and each machine will have information such as its IP address, the screen size and its position in the classroom. The username of the person using the machine is used in the GUI
but is never saved for privacy reasons. Its main purpose is for the teacher to know who is on the machine while using *Class Control* so the teacher knows who they are chatting with in the text chat box or who they are taking remote control of. The username won’t be added to the audit log or the database. The screen size for each machine is needed because *Class Control* was originally created for 17” Mac screens and nothing else. Now that *Class Control* is being developed for different screen sizes, it is important the server knows this information. The screen size is used for displaying screenshots of the screen correctly and for correct mouse movement coordinates for the remote control functionality. The classroom X and Y coordinates tell the GUI where on the grid of the classroom on the teacher’s screen this client should appear. The classrooms node also has its own name attribute so that the configuration database can have its own name that can be set by the teacher to whatever they want instead of just using a date timestamp.

### 7.4.2 Updating the Database

The database will be updated every time a new client is discovered. There will first be a check to see if the classroom the client is in exists. If it doesn’t, a classroom node will be created and the client will be added to it. If the classroom does exist, there is a check to see if the client already exists in it. If the client already exists, it is ignored, otherwise, the client will just be added to the classroom as it is new information.

### 7.4.3 Using the Database

The database will be used on the initial start-up *Class Control* to load previously saved information. This allows a teacher to not to wait for a scan to complete before choosing a classroom to connect to. The teacher will be
able to choose from multiple past known good configurations containing information that the server had discovered when a configuration was saved.

7.4.4 Multiple Databases

The last section mentioned that a teacher can choose from multiple configurations. When *Class Control* is terminated it will save the current state of the configuration into a file. It will check how many past configurations there are and will only keep the most recent configurations. The implementation chapter will talk in more detail about how many configuration files are kept and how they are stored.

7.5 Audit Trail

This log will keep track of connection attempts, whether successful or not, the time of which the program was started and quit and also, if possible, who was using the software. As the teacher and student perform different functions, their audit logs will differ slightly.

7.5.1 Teacher’s Audit Log

The teacher application will also have to keep track of password attempts to see if any unauthorized access was attempted. It will also keep track of when a broadcast was started and stopped and what machines were connected to during a session. It could also log chat messages and when remote control sessions took place.

7.5.2 Student’s Audit Log

The students log won’t contain as much information as the teacher’s but it will contain times of remote control sessions and broadcast sessions just like the teacher. It will also be able to track events such as being queried by a teacher for more information about the machine.
7.6 Extending Class Control

The previous sections give an overview of the final intended design of *Class Control* but consideration needs to be made about how the current state of the software can be adapted to allow this extra functionality.

7.6.1 Extending the teacher application

From the initial phase of this project there is already a piece of software called *Class Control* which is the main part of the software that allows a teacher to view a screenshot of each machine in the class it is connected to. This is displayed as a grid like the figure below:

![Figure 26: Thumbnail View Screenshot](image)

Each item in this grid relates to a machine in the grid layout of the classroom in St Andrews that this software was originally designed for. When a student enters the classroom, the client running on their machine will send a screenshot of their screen to the teacher’s server:
By using a button on the main screen GUI, the grid of thumbnails can also be switched to a ‘broadcast’ mode, pictured below:

Each one of the buttons in broadcast mode controls which machines in the classroom will receive the broadcast of the teacher’s screen. Each one can be individually toggled and there are buttons to turn on and off the broadcasts for groups of machines such as the front two rows only.

The main problem of this project is how to link the self-configuring and discovered information into Class Control so that this software can be used in any lab and perform the machine learning activities it needs to.
Currently, *Class Control* loads in the following way:

![Diagram](image)

**Figure 29: Old Class Control Life Cycle**

The teacher has to enter their password; they then see a loading screen, and then the main piece of software loads using the hardcoded list of clients that is stored in the source code. As the original project was only supposed to work in this lab, this was an acceptable way to run *Class Control*.

Now, the project requires new ways of loading a dynamic list of machines into the main software. This requires the software to scan the environment, as mentioned before, to discover what clients and classrooms exist and allow a teacher to choose from multiple previous configurations.

This process can be pictured in the diagram below:

![Diagram](image)

**Figure 30: Class Control Life Cycle**

It is clear that there are a lot new components to be added to the software. This section will give a brief overview of each component but the implementation chapter will go in to greater detail. There will be some
changes in the main software to adjust to using new information but these changes will also be explained in the implementation section.

**Scan or Load?**: When a teacher first loads the software, they are given the option to either scan the environment, explained later, or, if they know they have a good configuration saved, they can choose the load option.

**List DBs**: When a user chooses to load a previous configuration, there will be a screen displaying what previous configurations are currently stored. The teacher will be able to choose the one they knew last worked for them to save time in scanning the environment.

**Scan**: A scan will open up a socket to receive any messages from clients that are multicasting their information on the network as explained earlier. A scan is vital for the teacher to initially gather information about the classroom so if there are no previous configurations, a scan will be needed to get the first database set up. The scan is also performed the whole time the teacher is running so the server can learn if new machines appear in the current classroom to connect to them and so it can also learn information about the rest of the environment too.

**List Classrooms**: The next stage of the loading process will present the teacher with the list of classrooms that are in this database. The user can then choose which classroom they want to connect to, this classroom information will be loaded into the main teacher program so the connection to each client in the room can be made.

**Store Config**: The software will need to store the most recent version of any information it has gathered throughout its lifetime. When the
teacher closes the software, the teacher will be presented with a dialogue to give a name to the configuration file which can be shown later on the List DBs component.

**7.6.2 Adapting the student application**

The student application will also have to be adapted so that it can broadcast its presence to the environment. Section 7.1.2 shows a state diagram of the client software, the addition it needs is to sit in a loop broadcasting its availability while it is waiting for a teacher to connect to it. Once it receives a message to connect to it, it won’t need to broadcast that it is available anymore because only one teacher will be allowed to connect to this client at a time. This means, if the client is connected to a server and it receives another request to connect to a different server, it will ignore it and only accept any commands that come from the server it initially started a connection with.

**7.7 Naming Standards**

An image of the initial package tree can be shown below:

![Class Control Old Package Tree](image)

Each of these packages contains classes relevant to the package name. This stage of the project required new components to be added to this tree. The config package was
extended to contain a package called ‘ClassSetup’. It contains the implementation of
the new machine learning and configuration aspect of Class Control.
8. Implementation

This chapter of the paper will discuss how the various elements in the design section have been implemented.

8.1 Teacher Application (Server)

The teacher application is built up from various components that work together to provide the main functionality of Class Control. The following figure shows a screen shot of the current state of Class Control running on a machine in the graduate laboratory at James Madison University.

![Class Control Interoperability Screenshot](image)

Figure 32: Class Control Interoperability Screenshot

The screen shot shows the Class Control software running on a Windows machines with the client visible on the left running the Mac OS and the client on the right is running the Linux OS. This connection was set up without having to do an initial scan of the environment because the server had already gathered information about its environment in a previous run. How this works and the implementation of each of the components is now described in this chapter.
8.1.1 Network Scanning and Learning

When Class Control is loading, it can perform an initial scan of the environment that it currently sees. It will do this by opening a multicast port on port 58994 and address 238.255.0.1. These addresses are both changeable using the properties files explained in section 8.5. The server will then listen for any messages, for a period of 20 seconds, which a client is multicasting out to the network. The server is going to receive messages in the form:

“BC CLASSROOMNAME ROW COL TOTAL ROWPOS COLPOS SCREENX SCREENY”

As clients multicast their availability every 5 seconds, 20 seconds should be enough time for all, or at least most, of the available clients to have been recognized by the server. The server is also able to determine from the received message the IP address of where the message came from. The server can then access the current database it is using, check to see if the IP address already exists, if it does then update the values associated with the IP address, if not, it will add this machine to the database. The BC is just there so the server can check for this and know that this is the broadcast message because it may also be received while the main software is running.

The following figure shows that when a message is received, the information from it is added to the database:
The database can be thought of as a tree like the figure below shows:

![Database Tree Diagram]

Figure 33: Client Added to the Database

Figure 26 shows how new nodes can be added to the tree by going through the thought process of checking whether the new client is already in a classroom the database knows about then checking to see if this class already contains information about this new node. If the client already exists then nothing else needs to be done, otherwise the new client node is created and is added to either the existing classroom node or a newly created classroom node.
Figure 35: Adding a Client to the Database

This whole process happens every time a broadcast message is received so that the database will always contain as much information as possible. This new addition to Class Control requires a new set of classes to be added to the code structure of the project. The next figure shows these new classes:
This scan is able to be carried out throughout the whole runtime of *Class Control* so that the teacher is always looking for new machines to build up as much information in the database as possible about the environment. This is useful for when new clients join a classroom after the initial scan, the new clients can be displayed on the teacher’s GUI without the teacher having to do anything to refresh their screen. This connection process is explained more in section 8.1.4.

### 8.1.2 Database Selection

After the server has performed its initial scan, the teacher will be presented with a list of past databases to choose from. Each database will contain past configurations of *Class Control* that have been saved in the previous runs of the program.
The name of the database comes from an attribute tag that is stored in each database file. Section 8.1.2.2 shows how this tag is updated. It is read simply by grabbing the root node of the file as this is the classrooms node and then by looking at its name attribute. If a name wasn’t set then the time and date of the database’s last use will be displayed. The screenshot above shows a list of databases with a comment that has been entered by a teacher and one database that just has the time and date displayed because a teacher never entered a name for the configuration.

8.1.2.1 Recommending a Database

The teacher will not only be presented with a list of databases, but will also be given a recommended database that the server thinks is the most appropriate. It does by using a number of steps:

1. Perform initial scan of the environment
2. Compare this scan with previous databases already stored
3. Check each database for matching machines (Machines with same IP and Classroom name and position in the room)
4. Calculate how new the information is by calculating the number of machines that were found to still have the same information use this against the number of machines that were actually found to build a score for each database
5. Look at the broadcast option value for each client. For each client in the database that received the last broadcast add to the score if any of these clients are now visible
6. Choose the database with the highest score

These steps help Class Control offer a suggested previous configuration for the teacher to use to help speed up the process of connecting to a classroom because a database with the most up to date information is more likely to have information about other machines in a given classroom too.

When the configuration selection screen is about to be displayed, there is a scan for 10 seconds to give the server a peek at the environment to help it determine a configuration to select.

The literature reviews talks about a variety of machine learning techniques. This suggested database metric allows for a rudimentary approach to creating a machine learning algorithm to be used in Class Control, however, more advanced techniques such as Bayesian networks or case based reasoning could be used in future versions of the software.

8.1.2.2 Storing a Configuration

When Class Control is closed, the teacher will be presented with a small text entry box to give a name to the current configuration. If the current configuration already had a name, this will be displayed and the teacher will be able to modify it. If the box is empty, the date and time of Class Control closing will be used as the name of the configuration. The name of the database will be stored in the XML file so it can easily be retrieved. Section 8.1.2.3 explains why the filename doesn’t get changed.

The process of storing the most recent configurations is as follows:
**Step One:** In this example 4.xml is determined to be the current xml file in use. This is saved in the teacher’s configuration on the loading of *Class Control*.

1.xml  2.xml  3.xml  4.xml  5.xml

**Step Two:** The 4.xml is then moved into a temporary file and the original is deleted:

1.xml  2.xml  3.xml  temp.xml  5.xml

**Step Three:** All files before this file number are then renamed to be a file number greater. The colors in the image show the filenames changing but the file stays the same.

temp.xml  2.xml  3.xml  4.xml  5.xml

**Step Four:** When the files have been moved up one, there won’t be a file named 1.xml now so the temporary file can be renamed to this and we are left with five configurations in order of when they were last used.

1.xml  2.xml  3.xml  4.xml  5.xml

If a new scan was performed when *Class Control* was started, this is stored in a temporary file called 0.xml. When *Class Control* is shut down, there is a check for this 0.xml file. If it is 0 then *Class Control* pretends it was 5 and deletes the 5th oldest configuration to make room for the new one. Even if there are less than 5 configurations or even 0 previous configurations, this code will still work because there is a check to make sure the file exists before it is attempted to be renamed. If for example, there are only 2 previous configurations
then the 2\textsuperscript{nd} oldest will create a new 3.xml file. If there were 0 previous configurations, the new database is simply named 1.xml. The number of past configurations saved can easily be changed but for this project, 5 configurations were enough to make a proof of concept.

The image below shows a configuration selection on a trial run of Class Control:

![Figure 38: Configuration Selection 1](image1)

The figure shows that ‘test config 2’ has been suggested as the most appropriate configuration. This configuration could be chosen or any other configuration in the list could be. For this example, ‘test config 2’ will be selected. After running the main program and exiting it, Class Control is then started again. The configuration selection screen now looks like this:

![Figure 39: Configuration Selection 2](image2)
This is because the configuration was originally stored as 5.xml so it was last in the list. Now it is stored as 1.xml so it is displayed first.

8.1.2.2.1 Storing Broadcast Information

It has already been noted that while Class Control is running, any information gathered about clients in the environment will be stored in the database. Another thing stored in the database is a value to say whether or not the client had been set to receive a broadcast in the last session. When a teacher broadcasts to a class they are shown a panel like the one below:

![Broadcast Control Panel](image)

Figure 40: Broadcast Control Panel

Each client in the classroom has its name in this control with a black dot next to it. If a client is connected, this black dot will be red or green and is also a button. This button, when clicked will determine whether a client receives a broadcast or not. There is a row of buttons in this image from the original phase of Class Control that control various sections of the lab in St Andrews. These buttons perform actions such as only broadcasting to the two front rows of machines or only to the left half of the room etc. This means sections of the classroom can be left alone for students not part of the class or are doing their own work. These buttons were designed for the original lab but if a classroom has a grid layout, these buttons can be applied
to any lab. When, for example, the front two rows button is clicked, a message can be sent to every machine not on the front two rows telling it not to receive the broadcast. As each client has its row and column stored, this is an easy check for the server to make.

If these buttons don’t do what the teacher wants, if for example a classroom didn’t have a grid layout, a teacher could set the controls themselves by clicking each machine.

When Class Control is terminated there is a ‘BroadcastClientSingle’ array stored in the teacher’s configuration. This array contains whether a client is set to receive a broadcast or not so when Class Control is terminated, the value can be added to the client’s list of attributes. A teacher could then simply have a single classroom in five configurations but have each one set to take control of a different layout of the room.

8.1.2.3 Finding the Databases

In order for Class Control to display a list of databases to the user, it must be able to find and load multiple databases. This has been implemented by simply calling the databases 1.xml – 5.xml. This allows a loop to be created that can easily access each database instead of having 5 or more (depending on the number of databases) different statements. Multiple statements would still give the same functionality but less code makes the code look tidier.

8.1.3 Choosing a Classroom

Once a user has chosen a database to use or when a scan has been completed, the classrooms that are in this database are displayed to the teacher. The screen shot below shows the classroom selection screen:
The structure of the XML file helps retrieve the names of the classrooms in the file. If one thinks of the structure as a tree, the program simply looks at the classroom layer of the tree and returns each of the classroom names from each classroom node.

![Classroom Selection](image)

Figure 41: Classroom Selection

Each of the classroom elements contains a name attribute which is read by the class loading the Classroom Selection GUI and displayed in the list. If a teacher does not want one of these classrooms they can go back to the configuration selection screen or start a fresh scan.

![Selecting Classrooms from Database](image)

Figure 42: Selecting Classrooms from Database

8.1.4 Choosing Clients in a Classroom

When a classroom in the list is selected, the user can select the ‘View’ button. This brings up the panel shown below.
This allows a teacher to see what clients are stored for the selected classroom. The green square is part of the grid layout of the classroom. The green represents the currently selected client. In this example, there is no information for this client but the configuration says that there is one row of machines in this room with two columns. The blue block shows that there is information stored for that client. If the teacher clicks it, the green block will move on top of the blue block and the IP address of the machine will be shown. The figure below shows this:

You can also see where the green block used to be there is a gray block because there is no information about that client. If the teacher sees clients they do not want to connect to at this time, they can click delete and the client will be removed from the database. This could be done if the teacher knows this machine has been removed from the room and to save any attempt of a connection being made. Section 8.1.5 shows that a teacher may not want to make any new connections, if for example, they know the configuration only contains the clients they want.
8.1.5 Connecting to a Classroom

When a classroom is selected, the process of connecting to that classroom begins. To do this, the teacher’s application will grab the correct classroom node and begin to send a message to each of the children nodes as these will be the machines in the selected classroom.

The message will give the client a port number to start sending its screenshot images to. The server decides what this will be so that each client is given a different port number. The teacher’s property file contains a base port number to start from and this is incremented by one for each client. The screenshot below shows the messages sent to the two clients in the graduate lab at James Madison University.

```
Init Connection to:134.126.141.20 58998
Sending: stream59008 to 134.126.141.20 port: 58998
Sent
Init Connection to:134.126.14.52 58998
Sending: stream59001 to 134.126.14.52 port: 58998
Sent
```

Figure 45: Initialization Messages

While the server is connecting to each of the clients, the teacher is prompted to choose whether they want to receive any new connections. This means that if a teacher has a configuration that they know works and contains only the machines they’re interested in, they can disable the functionality of adding new machines that are in the selected classroom.

8.1.6 New Clients During Runtime

When the teacher loads a classroom from a previous configuration, there will be a chance that not all of the clients are running yet. To ensure that connections are made to these clients in the future, when the client eventually starts and begins broadcasting, the teacher will be scanning for these broadcast messages. This saves the teacher application from having to
constantly ping each client to see if they arrive. As the teacher is already
listening for information to build up the database, waiting for a received
availability message is a good way to solve this problem. First, the server
goes through the process of seeing if the received message from a classroom
already in the database, if it isn’t then there is a check to see if new
information is being stored or not. This is decided by the teacher after
choosing a classroom to connect to. If the teacher is storing new information
and this client is in a different classroom it is simply added to the database. If
the client is in the current classroom in use and is already in the database then
a connection is made to that client because this isn’t new information. If it is
a new client and the teacher doesn’t want to make new connections then it is
ignored. This ensures that the teacher’s selected machines will all connect
even if they aren’t all running at first when the teacher starts their software.

8.1.7 Dynamic Screen Refresh Rate

A new message was added to the protocol that allows the rate in which clients
send their screenshots to the teacher to change. The teacher can send a
message that says ‘refresh RATE’ where RATE is replace by the number of
seconds that should be between each screenshot update. As the server could
potentially have a large number of clients connected to it, each one of these
will be sending screen shot updates to the server. As more clients connect,
more images will be received and will have to be processed. To help reduce
this workload, every 10 clients the server connects to an extra second is added
to the refresh time. So for 50 computers, there would be a refresh time of 15
seconds set as the original value is 10 seconds.

8.1.8 Check for Client Running on Same Machine

As the server and the client use the same port numbers for some things, this
can be checked to see if they are already in use before Class Control is made
to load. The port numbers are set to be numbers that are very unlikely to be used by another program so if one is in use; a prompt can be displayed to the teacher asking them to close the client before the server is loaded up.

8.1.9 Audit Logs

One of the initial objectives for this project was to implement some auditing process in *Class Control*. This has been done in the form of a text file that is updated with messages sent to clients and messages received from clients. Each entry in the log contains a timestamp and the IP addresses of the involved machines. The text file is in plain text so it is easy to read by an auditor. The initialization messages from figure 46 show an example of an entry in the audit log.

8.2 Student Application (Client)

The student application also has also had some changes made to it to interact with the new capabilities of the teacher application. As previously mentioned, the changes have to be made while still maintaining the functionality that was possible in the first phase of the project.

8.2.1 Broadcasting Availability

The first change for the student is that it now needs to broadcast its availability to the environment so that a listening teacher application will be able to find it.

Every 5 seconds the application will broadcast a message revealing the client’s location and information about the machine. The structure of the message is as follows:

“CLASSROOMNAME ROW COL TOTAL ROWPOS COLPOS SCREENX SCREENY”

The various sections are:
**CLASSROOMNAME**: Name of the classroom the machine is in

**ROW**: Number of rows the classroom has for help with the GUI layout

**COL**: Number of columns the classroom has for help with the GUI layout

**TOTAL**: Total number of machines in case ROW & COL aren’t available

**ROWPOS**: Row number of client for placement in GUI

**COLPOS**: Column number of client for placement in GUI

**SCREENX**: Client’s screen width

**SCREENY**: Client’s screen height

It is worth noting that not all classrooms come with a grid layout, so a total number of machines figure allows an appropriate size GUI layout for the addition of all of the clients in the classroom. The screenshot below shows the output added to the audit log when multicasting to the teacher:

```
Multicasting out to Teacher...
Sending: GRADLAB 1 1 1 0 0 1280.0 1024.0
Multicasting out to Teacher...
Sending: GRADLAB 1 1 1 0 0 1280.0 1024.0
Multicasting out to Teacher...
Sending: GRADLAB 1 1 1 0 0 1280.0 1024.0
```

Figure 46: Broadcasting Output

From this figure, you can see that the name of the classroom is followed by the total number of rows and columns which are set to 1. The total number of machines in the class is 1 so the X and Y coordinates are 0,0. This is then followed by the screen’s dimensions. The dimensions are calculated by using the function `Toolkit.getDefaultToolkit().getScreenSize()`. This returns a Dimension object which I can then call getWidth and getHeight on to get the screen size of this client.

This information is all sent to the teacher’s application so that a layout of the environment can be built up and the teacher’s database will contain
information about classrooms, their sizes and information about the machines in the classroom.

8.2.2 Switching between connected and not connected states

There has to be a transition between broadcasting the client’s availability and being connected to the teacher. For this to happen, a message is sent to the client as part of the new protocol. This message is:

streamPORT

It lets the client know that the teacher is now ready to begin a connection and the client should stop broadcasting and start sending its screen shots to the teacher to begin its normal functionality in Class Control. The PORT part of the message is replaced by a port number for the client to start sending screen shot updates to. Section 8.3.3 explains how this port number is calculated.

When the teacher has finished its session with the client the ‘teacherbye’ message is sent to it. This message will tell the client to start broadcasting its availability again ready for the next connection from a teacher application.

8.2.3 Accept Commands Only From Teacher

When the message is sent from a teacher to begin the connection, the client is then only going to accept messages from this machine. This is to help reduce the chance of an adversary trying to take remote control while the teacher is in control. It doesn’t prevent an adversary taking control in the first place but the password to the teacher application should stop anyone that is unauthorized from using the teacher’s application.

8.2.4 Screenshot Sending Rate

If the client receives the message sent by the teacher saying refresh RATE, then a sleep thread is set to sleep for RATE amount of seconds between each
screenshot that is taken and sent to the server to help ease the load on not only the server but also the network.

8.3 Network Technologies

For this phase of the project there are a number of new messages being sent and received across the network. This section of the report looks at the transport mechanisms used and the reasons for doing so.

8.3.1 Multicast

Multicast messages are different from unicast messages because a unicast message is designed to send a packet to a single destination. A multicast destination will be sent to any machines in the same multicast group. If the teacher application is in a multicast group and clients send messages in the same group, the teacher will receive the message without the client having to know where exactly the teacher is. For this reason, multicasting was chosen to allow clients to broadcast their availability to an environment.

The figure above gives an example of a multicast set up showing how groups of machines can interact using a multicast routing protocol. This shows how...
machines in different classrooms could be able to communicate with each other without originally knowing where the teacher’s machine is.

**8.3.2 TCP**

TCP has been chosen to send control messages from one machine to another to ensure their delivery. TCP can guarantee this if both machines are connected.

**8.4.2.1 Control Message Server**

The student and the client will both have a control message server. This comes in the form of a thread that will always be running. The thread will open a socket to listen for messages and then, when one is received, the connection is set up, the message is received and processed, and then the connection is closed so that the socket can listen for the next message coming through.

Section 8.3.4 explains what actions the client and server take on new messages in the protocol.

**8.3.3 Port Number Allocation**

Class Control relies on various port numbers for the transportation of data to and from the teacher and students. These port numbers can be seen in the properties files of each of the teacher’s and the student’s applications.

![Properties File with Port Numbers](image)
More information on the properties file is explained in section 8.5. These port numbers were chosen because they are very large numbers and unlikely to be used by anything else. For receiving images a base port number of 56900 was chosen and incremented for each machine that connects.

### 8.3.4 Protocol Changes

The new messages that have been added to the protocol are the **streamPORT** message described in section 8.1.5. This message tells the client to move out the ‘broadcasting its availability’ state and into the ‘communication with the teacher’ state. The message also contains the port number for the client to send its screen shot images to as each client will need their own port number to communicate through. The original ‘teacherbye’ message is used to tell the client to disconnect from the teacher and resume broadcasting its availability.

The other addition to the protocol is:

“BC CLASSROOMNAME ROW COL TOTAL ROWPOS COLPOS SCREENX SCREENY”

As explained in section 8.2.1 this message is sent to the teacher from every client that is in broadcast availability mode. This is parsed and used by the teacher to add the client’s information to the database. The BC is used by the message checker class to determine what this message is and what the relevant action is.

Adding these messages to the protocol required very little addition to the message handling already in place. Further messages could still be added, if needed. The ‘refresh RATE’ message was also added to help provide the dynamic screen refresh rate ability. This message will make the client set the refresh rate of their screenshots to however many seconds the RATE says to.
8.4 XML Database

Section 7.5 discusses the addition of an XML database to Class Control. There are five databases used and each contains a previous configuration from one of the previous runs of Class Control. The final schema of the database is the following:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="classrooms">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="classroom" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="machine" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:attribute name="name" type="xs:string" use="required" use="required"/>
                    <xs:attribute name="rowpos" type="xs:positiveInteger" use="required"/>
                    <xs:attribute name="colpos" type="xs:positiveInteger" use="required"/>
                    <xs:attribute name="width" type="xs:decimal" use="required"/>
                    <xs:attribute name="height" type="xs:decimal" use="required"/>
                    <xs:attribute name="receive" type="xs:string" use="required"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 49: XML Schema
This can also be viewed in the simpler image below:

```xml
<Classrooms name (one)>
  <Classroom name total rows cols>
    <Machine ip rowpos colpos width height receive></Machine>
    <Machine ip rowpos colpos width height receive></Machine>
    ...
  </Classroom>
  <Classroom name total rows cols>
    ...
  </Classroom>
  ...
</Classrooms>
```

Figure 50: Simplified XML Schema

To access and create these files the org.w3c.dom library is used. This allows the creation of XML documents with a tree like structure where nodes can be searched for, modified and created using the tools available from this library.

### 8.4.1 XML Database Custom Methods

The methods provided by the org.w3c.dom library helped the creation of custom methods to retrieve or add information when appropriate. These methods are described below:

- **createXMLFile**: This method creates an xml document with the name passed into the function as the file name and then the classrooms node as the root node ready for classrooms is added to it.

- **addMachine**: This method allows for a single machine to be added to the database. Its parameters contain all the information the server needs to know about the client. The name of the classroom is also passed into this function. The method will check to see if the classroom exists that the client is from and will create a new classroom node if necessary.

- **getMachineList**: This method is used for when a classroom is selected. The server is able to call this method to retrieve all of the client information that has been stored for a given classroom.
**isClass:** This looks at each classroom node in the document to see if its name attribute matches the classroom that the server is looking for. It then returns a boolean result with its findings.

**getClassroomList:** This grabs all of the root’s children nodes as these are the classroom nodes. Each classroom node is then returned in a list that can be traversed to cycle through each of the classrooms. This is convenient for getting the name of each classroom to be displayed in classroom selection prompt.

**isMachine:** This function is used to check if a given machine is already in a classroom.

**deleteMachine:** This function checks the classroom for the given machine’s IP address and then removes it from the database.

These functions all help the server interact with the database and add and retrieve information when necessary.

### 8.5 Properties

*Class Control* comes with two properties files that contain information about ports and IP addresses used by the client and by the server. Figure 52 shows the teacher properties file.

![Properties File Screenshot](image)

This properties file can be edited after the software has been compiled meaning these values don’t have to be hardcoded and can be edited by an administrator of a lab if,
for example, one of these ports is already in use by another application. The properties file is read by the configuration class of each the client and the server when the software first loads so that the values in it can be used throughout the program.

8.6 Error Modes

Part of the adaptability functionality of Class Control is the ability to handle a loss of connection in the classroom when a teacher is connected. There are a number of ways this can be detected and handled. The three main methods of detecting and handling errors in a safe manner are described below

8.6.1 Thumbnail Click

When a client is connected, the screen shot being sent from the client is displayed to the teacher. The teacher can either click the thumbnail or one of the icons, shown below, and a message will be sent to the client. If the message fails to send, the screen shot will disappear and prevent the teacher from having any more interaction with the client.

![Error Handling Client Loss](image)

Figure 52: Error Handling Client Loss

If the client is there but a connection hasn’t been established yet, the teacher can also click on the empty thumbnail to force a connection to be made.

8.6.2 Student Disappears

As well as the thumbnail click causing the server to know if a client has been lost, every other message that is sent to the student is checked for errors too. These messages could be sent during the broadcast mode or during a remote control session. If the client is lost during a remote control session, the
remote control window is closed immediately to show that the client has been lost. The client will reappear in the grid layout on the GUI when it returns.

### 8.6.3 Teacher Disappears

It isn’t only the clients that could close unexpectedly; the teacher’s application could also disappear. If this happens while a client is connected, the client would carry on sending screenshots to the teacher. To help solve this problem, the client will also periodically ping the teacher every 15 seconds to make sure the teacher is there. If the client’s ping is unsuccessful, the client will stop sending its screenshot and return to its broadcasting mode.

### 8.7 Screen Size & Remote Control

There has been discussion in this report about how the size of the client’s screen is important and has to be stored in the database but there has been no mention yet of how it is actually used in the operation of *Class Control*.

![Remote Control Calculation](image)

Figure 53: Remote Control Calculation

The figure above shows the problem *Class Control* faces. The remote control window the teacher will see will be smaller than the actual size of the target screen. The coordinates of the mouse location can be grabbed from the single viewer on the left using event listeners. These coordinates then need to be transformed to coordinate relevant to the target screen.
To do this we use the size of the server’s single viewer window and use the target screen size in a calculation. The formula for the X coordinate is described below:

\[
\frac{\text{Server Window's X Coordinate}}{\text{Width of Server Window}} = \frac{\text{Target's X Coordinate}}{\text{Width of Target Window}}
\]

Figure 54: X Coordinate Calculation

The target x coordinate is calculated by first dividing the width of the server window by the width of the target window, giving us A. The x coordinate is retrieved from the server window and we perform x/A. This then gives us the target’s X coordinate. This has to be done each time the mouse is moved to update where the mouse should be pointing on the client’s screen. This allows a student sitting at the client to see exactly what the teacher is doing with their machine. Without this calculation, the remote viewing would still work but the remote control wouldn’t be correct. The y-coordinate is worked out in the same way but by using the heights of the window and target instead.

\[
\frac{\text{Server Window's Y Coordinate}}{\text{Height of Server Window}} = \frac{\text{Target's Y Coordinate}}{\text{Height of Target Window}}
\]

Figure 55: Y Coordinate Calculation

8.8 Installation

Class Control comes in the form of two .jar files that are created using Eclipse and can then be launched on any machine in the lab that can run Java applications. The .jar file can be made into an app on the Mac OS X using JarBundler to allow more control on the launching of it on runtime. Making Class Control an application file allows the addition of icons to make the software look more presentable.
A similar process can also be performed on Windows and Linux to create icons but as these are different operating systems, the process isn’t exactly the same.

8.8.1 Installing the Client

Installing the client is easiest if an administrator can access images of an operating system. The application can just be made to run at startup and if possible, be made to relaunch automatically if it is closed early.

The client’s properties can also be edited to show where in the classroom the machine is. This isn’t necessary for Class Control to function but it will make the GUI layout refer to the actual layout of the lab. Further work could be done to allow the GUI to be manipulated to allow a teacher to move thumbnails around to relate to the classroom set up. This is explained more in the future work section in the next chapter.

8.8.2 Installing the Server

Installing the server is simpler because it doesn’t require the classroom position to be entered. It is possible to change port numbers being used but the ones already set should suffice. The application will just be put on the target machine and run whenever required.
9. Evaluation and Critical Appraisal

This section looks at how the completed project performs. It will also look at how Class Control coped with testing and how it compares to the original objectives and requirements set at the start. There are also sections here to look at what possible future work could be done on Class Control in future work.

9.1 Testing

The testing of the new additions to Class Control, shown in Appendix A show that these have been implemented correctly and show that the core functionality required from the objectives has been satisfied. The self-configuration aspect has been tested and shown to update information in the database when appropriate. Class Control also has error modes that have been tested to ensure the software doesn’t fail if clients or the server goes down. The testing summary will explain how the tests have been carried out, their results and also the rationale for carrying out the test.

9.2 Comparison to Original Objectives

This section of the evaluation will look at how well the project has met the objectives that were set at the start of this project.

9.2.1 Primary Objectives

9.2.1.1 Self-Configuration

“Class Control should be able to configure itself in a class room with minimal input from the class administrator. The software will be able to discover what clients are in the classroom and find out information about the machines such as their location and screen size. It needs to do this so that it can be installed in new environments with ease. At the moment it is currently hard coded for the lab in St Andrews where it was developed”
Class Control is now able to discover what clients are in the classroom. The multicast mechanism to allow a client to broadcast its availability allows the server to discover them. An addition to the protocol means that more information about the client, such as screen size, can be sent to the server and added to a database. The properties files allow a class administrator to set the x and y coordinates for the machines, this is the only input needed from the administrator apart from actually putting the client and server on the machines.

9.2.1.2 Machine Learning

“Class Control will be able to learn about its environment in order to help speed up future uses of the software. For example, if Class Control learns the layout of a classroom, this information should be available the next time Class Control is run. The layout of the classroom would consist of the IP addresses of machines in the room and their screen sizes. The server would also learn if the classroom is arranged in a grid layout so the clients can be displayed to the teacher in a way that resembles their physical position in the room.”

Class Control can now use a database to store and retrieve information about clients it sees instead of having them hard coded somewhere. The additions to the GUI allow a teacher to either scan the environment or load from a previous configuration. Configurations can also be named so a teacher can personalize them. The schema of the database shows that enough information is stored about the machines and the classroom so
that if there is a grid layout, the machines can be placed in a grid on the GUI appropriately.

9.2.1.3 Self-Adaptability

“The GUI in Class Control will have to reflect these changes in the environment and also ensure that a client leaving doesn’t affect the operation of Class Control. Throughout the runtime of Class Control it is likely that clients will leave and join a classroom. This can be thought of as a teacher starting a lecture and students arriving late or having to leave early.”

The error modes that have been implemented in Class Control allow the server to handle the times when clients unexpectedly disconnect from the classroom. The clients also have error handling functionality if the server happens to unexpectedly disconnect. If a client is disconnected and the teacher tries to interact with it, its thumbnail image will disappear from the GUI.

If a client joins the classroom after the server has started, when the client begins to broadcast its availability, the server will see this, because it continues scanning the environment, and the client can be connected to the current session.

9.2.2 Secondary Objectives

9.2.2.1 Security

“Some form of security measure will have to be implemented to help prevent unauthorized connections to clients. There is potential here that an adversary could take advantage of a running client and pretend to be a teacher. This should not be allowed to happen because of the ability to view what is on a
student’s screen and the ability to take control of it. An audit log will also help show where the attack came from.”

*Class Control* has a password mechanism in place from the first phase of the project. This phase of the project added the ability for the teacher’s IP to be stored by the client; the client would then only allow access to its remote control and viewing features to this IP address. These features are a rudimentary measure at the moment but are satisfactory for the scope of this project. Section 9.5 talks about how further security measure could be added to *Class Control.*

### 9.2.2.2 Load Previous Configurations

“Objective 2.1.2 states that *Class Control* should remember any configuration that it ‘learns’ so that the information can be reused the next time the software is run. *Class Control* will also be able to load, not just the last known good configuration, but multiple configurations it has saved from previous runs of the software.”

In order to satisfy objective 2.1.2 *Class Control* has the ability to store five previous configurations that the teacher has used. These configurations can be displayed to the user at start up to allow them to choose a configuration that they decide is the one they want to use. The schema for the database allows the teacher to also name the configuration instead of just trying to remember the date it was made or the filename.
9.2.3 Tertiary Objectives

9.2.3.1 Audit Log

“Objective 2.2.1 mentioned the addition of an audit log to Class Control. This would be log file of any connections between the clients and the server and also when remote control connections are made. The audit log could also log chat conversations between the teacher and the students.”

The client and the server in Class Control already had diagnostic messages output to the console so now that the tool has been tested, these messages can be sent to an audit log. These logs keep track of IP addresses involved in communications and come with timestamps. This also helps improve the security and help meet the security objective.

9.2.3.2 Sub-systems in Classrooms

“A side project could be looked at to investigate how a single client in a classroom could be elected as the class leader and be responsible for telling the teacher’s application what machines are present in that room. If the teacher has the capability of receiving messages from any client in any classroom, this could result in more messages than the teacher could handle. Meeting this objective would help ease this burden. This could result in a lot more work than expected so had been placed as a tertiary objective.”

This objective was thought about but was never implemented. Clients could broadcast their availability to each other and collect each other’s information. An election could then be held...
to decide who multicasts to the teacher. Checks could be made to ensure there is always a client broadcasting outside of the classroom. A teacher in a different classroom would then see this machine’s multicast and the server could then request all of the information of a classroom from the elected machine. The multicasting messages functionality has already been implanted in *Class Control*, this system would just need an election algorithm in place for it to be implemented. Due to time and its low importance, this feature wasn’t implemented.

### 9.3 Extra Features

In addition to implementing the components needed to meet *Class Control*’s initial objectives an extra interesting feature was added to the project which came from an idea in the literature review.

A metric was created to allow *Class Control* to retrieve a suggested configuration depending on what it currently sees in the environment. Section 8.1.2.1 looks at this in more detail.

Storing the broadcasting state was also added to help complement the machine learning aspect of this project and also add to the score that is calculated for suggesting a database to a teacher.

The original *Class Control* software was created for an environment that had one teacher machine in front 70 machines set up for students to use. This project created the ability for the teacher to use any machine in the room as the main computer.

### 9.4 Comparison to Requirements

Apart from not implementing sub-systems in classrooms, all of the requirements from the requirements document have been met. This document was created to
provide a list of features that had to be implemented in order to satisfy the original objectives. Below is the list of requirements and a note on if they were met or not.

<table>
<thead>
<tr>
<th>Req. #</th>
<th>Requirement</th>
<th>Importance</th>
<th>Passed/Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Self-Discovery of classrooms and machines in them</td>
<td>High</td>
<td>PASSED</td>
</tr>
<tr>
<td>2</td>
<td>Clients must make themselves known to the server</td>
<td>High</td>
<td>PASSED</td>
</tr>
<tr>
<td>3</td>
<td>User is to be presented with a GUI with will display the layout of the environment</td>
<td>High</td>
<td>PASSED</td>
</tr>
<tr>
<td>4</td>
<td>Keep full existing functionality of Class Control</td>
<td>High</td>
<td>PASSED</td>
</tr>
<tr>
<td>5</td>
<td>The system will not store information about students</td>
<td>Medium</td>
<td>PASSED</td>
</tr>
<tr>
<td>6</td>
<td>The system will be scalable so it can still work in classroom of sizes of approximately 70 machines like it did in the first phase of this project.</td>
<td>Low</td>
<td>PASSED</td>
</tr>
<tr>
<td>7</td>
<td>The system will be able to run on any OS that can run Java</td>
<td>High</td>
<td>PASSED</td>
</tr>
<tr>
<td>8</td>
<td>The system will be able to handle unexpected errors</td>
<td>High</td>
<td>PASSED</td>
</tr>
<tr>
<td>9</td>
<td>Dynamic Screen Refresh Rate</td>
<td>Medium</td>
<td>PASSED</td>
</tr>
<tr>
<td>10</td>
<td>Sub-systems will collect information about the classroom they are in</td>
<td>Low</td>
<td>FAILED</td>
</tr>
<tr>
<td>11</td>
<td>The system will keep an audit trail</td>
<td>Low</td>
<td>PASSED</td>
</tr>
<tr>
<td>12</td>
<td>Server must store previous configuration states for use in future use of the system</td>
<td>Medium</td>
<td>PASSED</td>
</tr>
<tr>
<td>13</td>
<td>Server must be able to handle clients joining and leaving the environment</td>
<td>High</td>
<td>PASSED</td>
</tr>
<tr>
<td>14</td>
<td>Information about classrooms and machines in them will be stored in XML format</td>
<td>High</td>
<td>PASSED</td>
</tr>
<tr>
<td>15</td>
<td>Existing protocols of Class Control must be adapted to ensure new information needed by the server about clients is interpreted correctly</td>
<td>High</td>
<td>PASSED</td>
</tr>
</tbody>
</table>

Table 1: Passed Requirements

This table shows that 14/15 requirements were met. Only a low priority requirement wasn’t satisfied but this was related to a tertiary objective and doesn’t affect the success of this project.
9.5 Future Enhancements

As this project shows, Class Control has been created to be an extensible piece of software. The modular code layout allows for new components to be added or existing ones to be easily worked on.

Although this project has added a completely new level functionality to Class Control this section of the paper will look at some other areas Class Control could be improved in to not only increase performance but also increase its functionality in the classroom.

9.5.1 Image Difference vs. Whole Image

The main purpose of Class Control is to give the teacher the ability to broadcast their screen to a classroom of computers. As presentations often have the same slide open for more than a few seconds and up to a few minutes, the image sending could be optimized to only send an image when it changes from the previous screen shot taken.

This could also be improved further so that only the differences from the previous image are sent and not the whole image. This wouldn’t work as well on presentations but would be better for remote control sessions when the whole screen isn’t changing.

9.5.2 GUI Update

The GUI in class control provides all the functionality required for the teacher to use Class Control to its full potential. However, work could be done to improve the look of the software to make it look more professional.

More work could also be done to allow a teacher to customize the GUI and make more changes to the layout of the clients. A teacher could be permitted to move around single thumbnails so that they relate to the
layout of the class if layout information isn’t available. This could then be extra information that could be stored in the database for future runs of the software. The broadcast screen could also have customizable buttons or settings that the teacher could create while running Class Control and then save them for future use.

9.5.3 File Transfer

The ability for a teacher to broadcast a file to the clients would be a good addition to Class Control. A teacher could broadcast exercises or homework tasks to the class instead of printing them or uploaded them to an online server. The homework would be on the student’s machine and ready to start working on. Clients could also then receive the ability to send files back to the teacher such as completed homework.

9.5.4 Sub-systems in Classrooms

The sub-system in classrooms feature could be implemented as a future project to save a teacher having to interact with a whole school of machines. This would help increase performance for the teacher and speed up the environment information gathering.

9.5.5 Control Machines from Multiple Classrooms at Once

Currently, Class Control only permits a teacher to take control and view machines in one classroom only. A future update could allow a teacher to select machines from multiple classrooms to view or take control of. An example of this usage could be a principle wanting to see the teachers’ machines in each classroom to ensure that the teachers are using them for work and class time.
9.5.6 Saving Conversations

The way the database has been created allows for more things to be added to it by simply changing the schema a little. This means that for each client that is stored in the database, a conversation history could also be stored. The conversation history would a teacher and student to resume a conversation from either a previous session or to resume a conversation if the client or server crashed. This could also raise some security concerns so care would need to be taken with this future work idea.

9.5.7 Security

The discussion in section 9.2.2.1 shows that more work is needed to improve the security of Class Control, especially if it is ever turned into a commercial product. Encryption could be a key part of the extra security. Images and remote control commands could be encrypted so that no one else could listen in on the traffic being sent between the clients and the server. A symmetric key could be used between the server and client to allow for quick encryption and decryption, especially of screen images.
10. Conclusions

This final chapter takes a look at what went well in the project and what didn’t and also features a personal summary from the author about this project.

10.1 Achievements

The completion of this thesis project achieved a lot in extending the functionality of *Class Control*. The main achievements are described below

10.1.2 Extending the Original Project

The original *Class Control* was highly successful and went on to win an award in Scotland’s Young Software Engineer of the Year award 2011. The original tool, however, was only created to run in the lab in St Andrews so a lot of information about the clients it would be connecting to was hard coded. Thanks to the modular layout of the code in the first project, it was easy to add the extra functionality required to satisfy the objectives of this project. Creating a project that not only functions correctly but can also be further extended and updated is a great achievement in itself.

10.1.3 Machine Learning

One of the main goals of this project was for *Class Control* to learn about its environment so that any information gathered would benefit future runs of the software. This was implemented in the form of an XML database with multiple configurations. *Class Control* is able to provide a teacher with multiple loading options. The user can either perform a new scan or load a previous configuration that contains information from a previous session of *Class Control*. 
10.1.4 Self Configuration

The other main goal of Class Control was to add the ability for the server and clients to interact with each other by themselves and without any hard coded IP addresses. This was implemented as a multicast broadcast and the server is able to discover information about the classroom without any input from a class administrator, unless the grid layout is needed. The server is able to join the multicast group and, thanks to an update to the protocol, receive messages from any clients that are running.

10.2 Lessons Learned

The main lesson learned from this project is the importance of a literature review and how it can help inspire ideas for the development of the project. This first phase of this project was set by the original supervisor so there wasn’t much room for different ways of tackling the problem because he knew what he wanted and how to go about doing it, for the most part.

This projects literature review helped inspire things such as creating a metric for a suggested database. It also helped inspire the idea of using a database to store information for future runs of the software. Without the literature review it would have been a lot more difficult to meet the project’s objectives.

Time management is also another key skill that has been enforced during this project’s completion. A time plan was made at the start of this project but didn’t account for external activities affecting it such as homework from other classes.

10.3 Personal Summary

I have enjoyed working on this project again and I am very happy I was able to jump straight back into the code and begin adding to it. The modular layout of the code and splitting it all up in the original project helped greatly with this process. It has been very interesting learning about different methods that machines can use to
learn about not only their environments but other applications of it too such as playing games.

I would definitely consider carrying on this project in my own time to add some of the extra functionality that was mentioned in the future work section. It was very rewarding to see Class Control working in the lab here at James Madison University without having to hard code the IP addresses of the machines. I think this project was a great success and I look forward to discussing it with my peers.
Appendices

Appendix A – Testing Summary

This section contains a summary of the tests performed on Class Control to ensure that the new components integrated successfully with Class Control. Each test comes with an explanation of the test, the expected outcome, the actual outcome and rationale as for why this test was needed.

A.1 Server Tests

The server and the client are two different pieces of software so they have been split up in this testing summary. The following tests refer to the server application.

A.1.1 Environment Scan

**Test:** The server must be able to listen for clients and receive messages from them.

**Rationale:** The server must be able to do this so that information in these messages can be added to the configuration database.

**Expected Outcome:** A string will be displayed by the server in the console window to show that a message has been received and this will also show the contents of the message.

**Actual Outcome:** A string for each received message is displayed in the console window

**Result:** PASS

A.1.2 Database Update from Environment Scan

**Test:** When a message is received it must be parsed and added to the database
**Rationale:** The database stores information about all of the clients that is discovered during a scan. Each message received contains information about the client that is used by the main program of *Class Control*.

**Expected Outcome:** When a message is received it must be parsed and converted to an XML format and added to the database

**Actual Outcome:** The database reveals the message has been converted to an XML format and been added to it.

**Result:** PASS

**A.1.2.1 Ignore Duplicate Information**

**Test:** There should only be one entry per client in the database so if a message is received that contains information it already has, then it should be ignored.

**Rationale:** The server contains to all of the clients contained in the database in a given classroom. The server should not be allowed to connect to the same machine multiple times.

**Expected Outcome:** Multiple messages with the same information will be received by the teacher but only one entry will appear in the database.

**Actual Outcome:** Only one entry appears in the database.

**Result:** PASS

**A.1.3 Correct Classrooms Displayed from Scan**

**Test:** When a scan completes it should the teacher a list of classrooms for the teacher to choose from. This test ensure the full list of classrooms is displayed.
Rationale: The teacher may want to connect to a classroom that isn’t the one they are currently in. All classrooms detected in the scan must be displayed to the teacher.

Expected Outcome: All classrooms from the database will be displayed in this list. A text entry saying the name of the class confirms the sever has found the classroom.

Actual Outcome: All classroom names from a given database are displayed

Result: PASS

A.1.4 Loading Previous Configurations

Test: There are up to five precious configurations that can be displayed for the teacher to choose from. Each one is named 1.xml-5.xml. The server must check how many of these five exist and then display them.

Rationale: The machine learning aspect of Class Control allows a teacher to select from previous configurations.

Expected Outcome: When five databases are stored, all five should be displayed to the teacher on the configuration selection screen.

Actual Outcome: All five are displayed to the teacher.

Result: PASS

A.1.4.1 Loading Name of Previous Configurations

Test: Instead of displaying the filename of the database the teacher will be displayed a name of the database which is set by the teacher when they exit Class Control.
**Rationale:** The filename of the database x.xml doesn’t give the teacher any information about the information stored in that configuration.

**Expected Outcome:** The configuration selection screen will display the names of the configurations and not their filenames.

**Actual Outcome:** The configuration names are displayed successfully.

**Result:** PASS

**A.1.5 Classrooms Displayed from Selected Configuration**

**Test:** This test is similar to A.1.3 except a previously stored database is now used to display a list of classrooms.

**Rationale:** The teacher may have a previous configuration stored that contains information about other classrooms.

**Expected Outcome:** Classrooms should be displayed in the same way as test A.1.3.

**Actual Outcome:** All classrooms from the configuration are displayed

**Result:** PASS

**A.1.6 Edit Classroom Screen Displays Classroom’s Clients**

**Test:** A teacher may want to view what clients are in the database for a given database.

**Rationale:** The teacher should be given as much opportunity as possible to control what clients they connect to. This edit screen allows the teacher to view what clients they are going to connect to before actually connecting to them.
Expected Outcome: The GUI should display a grid layout of the classroom and display information about clients there is information about in the database.

Actual Outcome: Client information is displayed to the teacher

Result: PASS

A.1.6.1 Clients Information Deleted from Classroom Edit

Test: A teacher should be able to delete a client from a classroom and then its information will be removed from the database.

Rationale: A teacher may not want to connect to a client that is in a previous configuration or was picked up during a scan.

Expected Outcome: when the delete button is clicked, the client’s information will be removed from the database.

Actual Outcome: The client’s information is removed from the database.

Result: PASS

A.1.7 Clients Assigned Unique Port Numbers

Test: As the server initializes connections to each the clients it must assign a unique port number to each one

Rationale: The server receives screen shot updates from each of the clients. To do this it creates a connection on a separate port for each client. Two clients wouldn’t be able to connect the same port on the server.

Expected Outcome: A base number is incremented by one for each client connected to.
**Actual Outcome:** Console output shows that the base port number has increased by one and given to each client to connect to.

**Result:** PASS

**A.1.8 Client Unexpectedly Leaves**

**Test:** When the server is running, if a client unexpectedly leaves, there should be no future communication with this client until it returns, the client will also be removed from the GUI

**Rationale:** If there is no client there to communicate with, it will cause errors with messages that are trying to be sent. By stopping all future communication, it will prevent these errors from occurring.

**Expected Outcome:** When the ‘take control’ button is clicked after a client has disappeared, the thumbnail for the client should disappear to show the client has left.

**Actual Outcome:** The thumbnail disappears as expected

**Result:** PASS

**A.1.9 New Client Joins Session after Main Program Starts with Teacher’s Permission**

**Test:** If the teacher wishes to accept new clients connecting to the current session that aren’t in the current configuration then these clients will be added to the GUI when they appear.

**Rationale:** The initial scan or configuration may not have picked up information about the whole class yet. This allows the clients to be added to the session if they arrive while Class Control is running.
**Expected Outcome:** The GUI will update when new clients arrive in the classroom and the database will be updated with the new client’s information.

**Actual Outcome:** The GUI is updated and the teacher will see screenshots being received from the new clients and database contains client’s information.

**Result:** PASS

**A.1.10 Client is blocked after Main Program Starts because of Teacher’s Permission**

**Test:** If a teacher has selected that new clients can’t join the session then nothing will happen if a new client is started in the same classroom.

**Rationale:** The teacher may have a configuration saved that contains only the clients they want to connect to.

**Expected Outcome:** A client will start in the classroom but nothing will happen to the GUI or database

**Actual Outcome:** Nothing happens

**Result:** PASS

**A.1.11 Configuration Panel Name Edit Window Appears on Exit**

**Test:** When the teacher decides to quit Class Control a panel will be shown to the teacher containing the name of the current configuration in use or the current date if one isn’t set.

**Rationale:** The teacher needs an option to give the configuration a meaningful name if they wish to use it again in the future

**Expected Outcome:** Class Control will close but a window will appear containing the name of current configuration.
Actual Outcome: The configuration save window is displayed

Result: PASS

A.1.12 Configurations Reorder Themselves

Test: If, for example, the database 5.xml is used it will need to be stored as 1.xml so that when it is displayed in the next run of Class Control, it will be displayed at the top of the list.

Rationale: Configurations are to be displayed to the teacher in order of when they were last access. This allows the most recently used configuration to be displayed first.

Expected Outcome: The name of 5.xml will be displayed at the top of the configurations list the next time Class Control is run.

Actual Outcome: The name is successfully displayed at the top of the list.

Result: PASS

A.1.13 Configuration Name Updated

Test: If a teacher changes a name of a configuration, this should be displayed the next time Class Control is run on the configuration selection screen.

Rationale: A teacher may want to change the name of a configuration to make it more meaningful.

Expected Outcome: The new name will appear on the configuration selection screen the next time it is run.

Actual Outcome: The new name appears where expected.

Result: PASS
A.1.14 Fresh Scan is Saved as a Configuration

**Test:** The new scan is saved in a temporary configuration called 0.xml, when Class Control closes it should be stored as 1.xml.

**Rationale:** The scan creates a new database and as it is the most recently used databases it will need to be stored in 1.xml for the same reason as test A.1.12.

**Expected Outcome:** The new database will be stored in 1.xml. If 5.xml exists, it will be deleted and replaced by 4.xml.

**Actual Outcome:** Database is stored successfully and the old 5.xml is deleted.

**Result:** PASS

A.1.15 Audit Log is Updated

**Test:** As the server interacts with clients, log messages should be added to an audit log containing interaction details such as the IP addresses involved. This test is also used for testing the client’s audit log as the implementation is the same.

**Rationale:** The audit log can be used for security purposes at a time when someone needs to investigate what clients were accessed by the server at a given time.

**Expected Outcome:** When a connection to a client is initialized the audit log should contain an entry about this.

**Actual Outcome:** Audit log is updated with new entry.

**Result:** PASS
A.1.16 Suggested Configuration is Selected

**Test:** The server has a metric that will select a suggested configuration to the teacher depending on what it sees in a quick scan of the environment.

**Rationale:** The machine learning aspect of the project requires the server to perform better after each run. If a teacher can pick a suggested configuration, this means the server has calculated this will be the quickest and possibly most efficient configuration to connect to.

**Expected Outcome:** The configuration containing information about currently running clients such be selected as the suggested configuration.

**Actual Outcome:** The correct configuration is selected.

**Result:** PASS

A.2 Client Tests

The following tests show what tests were performed on the client software. The client doesn’t contain the same level of new functionality as the server but there is still some new functionality that needs to be tested.

A.2.1 Message Multicast

**Test:** The client must broadcast its availability to the environment.

**Rationale:** The server needs to be able to discover what clients are available in order to be able to connect to them.

**Expected Outcome:** A console output will show that a message has been sent to the multicast group.

**Actual Outcome:** The console output confirms that a message has been multicast.

**Result:** PASS
A.2.2 Connecting to Server

Test: The client must be able to connect to the server when the server is ready to initialize a connection.

Rationale: The client has to connect to the server to begin the main purpose of Class Control for the remote viewing and control.

Expected Outcome: When the client receives the stream message, a connection must be made to the received port number and then screen shot sending will commence.

Actual Outcome: Screen shot sending commences and the server can see the client’s screen shot updates

Result: PASS

A.2.3 Switching Between States

Test: When the teacher has finished with the client, the teacher will send a goodbye message. The client should then resume broadcast its availability.

Rationale: The client should always be running and always be ready to accept a new connection whenever the teacher has finished with it.

Expected Outcome: Client will resume multicasting availability when teacher leaves.

Actual Outcome: Client returns to multicasting its availability as expected.

Result: PASS

A.2.5 Teacher Unexpectedly Disconnects

Test: If the teacher disappears, the client should detect this and resume its availability multicast.
**Rationale**: The server may unexpectedly crash so the client needs to handle this so it doesn’t try to communicate with it. When the server resumes, the connection can be reestablished because the client will be back to multicasting its availability.

**Expected Outcome**: Client detects teacher has gone using its ping mechanism and returns to availability broadcasting.

**Actual Outcome**: Client returns to availability broadcasting as expected.

**Result**: PASS

### A.2.6 Screenshot Refresh Rate Change

**Test**: Client’s screenshot updates will change depending on what the server requires.

**Rationale**: The server receives screen shots from every client it is connected to. This can be a burden on the network or the server if this is a lot.

**Expected Outcome**: Screen shots are taken at intervals to match refresh rate set by the server.

**Actual Outcome**: Screen shot rate changes.

**Result**: PASS
Appendix B – User Manual

This appendix contains a simple user manual to guide a new user through the new additions to Class Control. Each phase has been broken down into the following sections.

B.1 Original Loading Screen

When Class Control first starts up, the original loading screen is displayed. It is preceded by a password entry box. The default password is currently teacher. This is encrypted and compared to an encrypted key in the server’s configuration file.

Figure 57: Password Entry Screen

Once the correct password has been entered, the loading screen appears:

Figure 58: Class Control Loading Screen

B.2 Scan or Load

When the loading screen has cleared the first new addition to Class Control appears.
The scan or load panel allows the teacher to select whether they would like to either scan the environment to build up a new configuration or they can choose load to select from a previous configuration.

**B.2.1 Load Screen**

If the load screen was selected then the teacher is presented with a panel that allows them to select from previous configurations. A suggested configuration is also displayed by the ‘(suggested)’ label. This list contains the names of previous configurations that have been saved by teachers typing in a name for them when they quit the program.

**B.2.2 Scanning**

If a scan was chosen to be performed, then this screen is skipped because a new configuration is being created.

**B.3 Classroom Selection**

The teacher is then able to choose a classroom to connect to. If a classroom doesn’t see a classroom they want to select then they have option to perform another scan. If a teacher wants to view the contents of a classroom they can click view to see the
window shown in B.3.1. Otherwise, a teacher can select a classroom and click “choose” to begin the main program.

![Classroom Selection](image)

Figure 61: Classroom Selection

**B.3.1 Classroom Editing**

A teacher may want to view what machines they are connecting to before loading the main program. By clicking ‘view’ the teacher is able to view what clients are stored in the database.

![Classroom Editing Screen 1](image)

Figure 62: Classroom Editing Screen 1

The example classroom above contains two machines that are shown by the small grid on the right. The green block shows the currently selected machine and the text box shows there is no information for this client. The blue box shows there is some information stored about this client so clicking it moves the green box on top of it as shown below:
The previous location of the green box is grey because there is no client information. Now, however, there is an IP address in the text box showing information about this client. If the teacher doesn’t want to connect to this client for some reason, they can click delete and the client will be removed from the database.

If the teacher is happy with the classroom information and they click ‘choose’ a panel will be displayed asking if the teacher wants to accept new connections. If the teacher just wants to stick to the configuration and nothing else, then clicking no will keep this. If the teacher wants all of the clients in a classroom that aren’t in the database yet to connect to the session then clicking ‘yes’ will allow this.

**B.4 Main Program**

The main program loads as normal and connects to the clients that are stored in the database. The normal operation of *Class Control* now resumes.
B.5 Exiting Class Control

When the teacher has decided they are finished with *Class Control* they will be displayed with a small text box containing the name of their current configuration. If a name hasn’t been set yet, like the example below, the teacher can enter a new name that relates to their session or can remind the teacher what configuration this was. This is entirely optional, just the date can be left as the configuration name if the teacher desires.

![Closing Screen](image)

**Figure 65: Closing Screen**

Whatever name is set will then be displayed in the load screen from section B.2.1. Clicking OK, will save the name and *Class Control* will exit.
Appendix C – Status Report

Overview:

The software came with the following functionality:

- Screen replication that can either be broadcast to the entire class or sent on a one to one connection from student to teacher.
- A broadcast can be received by a client and shown in full screen mode to the user and block any keyboard or mouse input
- A grid of screenshot images of the entire class can be seen by the teacher
- Ability to view a larger image of each client’s screen and be able to remotely control this computer and also broadcast this screen to the rest of the class.
- Whether or not a computer receives a broadcast can be controlled by the teacher.
  Groups of computers can also be controlled in the same way.
- The teacher can hide their screen from the class if they want to keep full screen control but not reveal what they are doing.
- Security features such as encrypted user authentication to prevent unauthorised access.
- Chat feature to allow teacher and students to communicate. Also comes with ability to change availability status

It now comes with the following functionality:

- Ability to scan an environment for running clients
- Connect to a classroom and still allow new clients to join the session
- Store configurations of current clients in the classroom and whether or not they’re set to receive the image broadcast or not
- Give the teacher the ability to edit classroom configurations
- Give the teacher the ability to stop new clients joining the session
- Configurations can be saved with a name relevant to the teacher or classroom
- The server can suggest a possible configuration to select
- The teacher can choose from previous configurations that were saved in previous runs of the software

Software:

The software for this project comes in the form of two executable applications. One application is for the teacher and the other is to be installed on each of the students’ (client) computers. Each comes with an editable properties file.

User Manual:

A full user manual for using the new functionality has been placed in Appendix B of the report.
Appendix D - Interoperability

James Madison University Graduate Lab

*Class Control* was originally intended to be run in the Mac OS X lab in St Andrews but thanks to using Java as the only language for this tool, it opens up the ability of letting *Class Control* work on multiple operating systems with no changes to the code as there is no native code used at all. The image below shows three machines in the lab all running three different operating systems: Mac OS X, Windows Vista and Ubuntu/Linux. The machines either side are running the client software and the machine in the middle is running the server program.

This next image shows that when I click the broadcast button, the Windows Vista screen is now displayed on each of the client machines. This required no change to the *Class Control* code apart from the new machine learning and self-configuration additions.
This following image shows the other part of Class Control’s main functionality, viewing the client’s screens.

Bibliography


http://download.oracle.com/javase/1.4.2/docs/api/java/awt/Robot.html


http://www.abconsulting.com/html/Tutor_Control.html


http://www.apple.com/remotedesktop/


