Contents

Chapter 1

Background

Computer Based Assessment (CBA) and Comptuer Aided Assessment (CAA) are not new concepts with large amounts of research having been done in the area. However, there are still concepts contained within this branding that have not yet been covered. Certain realms under the automated marking banner have been researched extensively. One such area that has been almost bypassed is the automated marking of Graphical User Interfaces (GUIs).

This chapter focuses on CBA in general, the teaching of programming, assessment and the use of CBA and non CBA systems both educationally and commercially.

1.1 CAA/CBA

Increases in the numbers attending university[1] have caused rethinks into the way courses are delivered and assessed. Changes were required to provide more efficent methods of course delivery and assessment. Computers were the obvious choice to automate certain parts of the assessment process. Numerous reasons [MMM04] have been suggested on why academics want to use computers to assist their course. These reasons include:

- To increase the frequency of assessment
- To encourage students to practice their skills
- To increase feeback to students and lecturers
- To increase consistency
- To reduce the marking burdon

Such changes, as well as bringing about the aforementioned benefits, do place more emphasis on students to take their learning into their own hands. This does not always go unopposed. Students are often reluctant to make alterations to their accustomed learning methods [AT95] even though computer based assessment has been shown to improve performance in summative assessments [CE98a].

[1]The most recent university entries have shown a reduction on several courses, but numbers are still signigicantly up on what they were 10 years ago.

Computers have become such an intrinsic part of assessment that their usage has been split into two main categories. Computer Assisted Assessment (CAA) refers to the use of computers in some stage of the assessment process, whereas Computer Based Assessment (CBA) refers to the use of computers in all stages of assessment from delivery of the question to provision of feedback [CE98b].

1.2 Teaching Programming

The teaching of programming is an important field in many disciplines throughout the academic spectrum. The learning of any language, programming or spoken, is a hands-on process and requires practice. Over the years, the methods used to teach programming to students have undergone several revisions. Reasons for this are numerable, many brought about by the demands being placed upon current programmers. The inception of the Graphical User Interface (GUI) and the advancement of programming languages have altered the skill sets employers are looking for. No longer is it enough to just create a working program, the program must now be efficiently designed and be both aesthetically pleasing and ergonomic. However, it is not merely the methods that have been changed, but also the languages being taught. Today's object oriented programming languages such as C++ [Str00] and Java [Mic05] provide both power and flexibility in a high level environment. Thus making them obvious candidates for teaching to students.

1.2.1 General Teaching Practices

In order for students to become all round competent programmers, there are several distinct parts to teaching programming that need to be covered. Each topic is equally important, lacking the knowledge in one domain will significantly impede a student's programming ability. For explanation purposes, the sphere of programming has been broken down into three distinct sections.

Syntax and Fundamentals

Before being able to write a working program, it is self-evident that a programmer

must first learn the syntax and the fundamentals of the language in question. Returning to our foreign language metaphor, you cannot write a sentence without knowing how it is to be structured and how the words are spelt.

The syntax needs to be tackled early on in a programming course. It is an intrinsic part of learning to program, without it students will not be able to create a program that compiles. The syntax explains how to create legal sentences or in this instance, lines of code. Once completed, the semantics can be covered.

The fundamentals of a language should cover everything from classes to variables. There are two schools of thought about how Object Oriented (OO) languages should be taught. The first is to teach it as a procedural language at the beginning, covering all aspects e.g. data types, conditionals, functions / methods before then explaining the concepts of objects. By the time the students have a grasp of the concepts they will have need to create a functional objects. The second is to start off teaching Object Orientation and then covering all other topics as they are required in the explanation of the OO programs. It has up to now not been proven that one approach yields better results and both approaches are used.

Testing and Debugging

The benefits of regular testing during the development of a computer program should be apparent. However, the number of students[2] who think it reasonable to submit a solution without a thought for testing is astounding. For this reason, it is a practice best nurtured at an earlier stage of programmer development in order to be ingrained in the programmer's subconcious. It is not the case that experienced programmers are unable to be taught to test and debug, they just need the desire to change [dJ05]. To have become an experienced programmers it surely follows that high quality, working programs must have been produced. Without testing, such programs are unobtainable.

Testing is a two fold process and must be taught in conjunction with debugging. Debugging is the process of analysing faulty code in an attempt to locate the errors contained within. Debugging is a skill which programmers develop by making mis-

[2]Based on experience as a demonstrator in a first year Java programming module

takes in the first instance and being able to recognise when they have recreated them. Novice programmers have not yet gained this experience and therefore need to be taught the basic methods for error identification and correction. One such technique is binary chop which involves commenting out sections of source code until the block containing the error is located. These techniques, however simplistic, form the basis of the methods they will use later in their programming careers.

Design and Layout

There are several typograpical conventions as far as programming is concerned. These recommendations include the length of identifiers, the amount of white space and the indentation of the source code. The reasoning behind them is simple, readability. Tidy source code is an order of magnitude easier to debug than code put together in a haphazard fashion. Making source code more readable, will allow programmers new to a project to understand it's makeup more readily. Thus reducing the time needed for acclimatisation.

Whilst important, the layout only helps us read what has been written, the design of the program should be a better indication of what it is attempting to do. The hardest programming skill to master is that of design. Large numbers of books have been written on this topic, the most famous of which being Design Patterns by the 'gang of four' [GHJV95]. Students must first be taught the basics of design before attempting to comprehend such topics as frameworks and patterns. To give students a good foundations to proceed from, all areas of Object Oriented design need to be covered, from procedural methods, through classes, to inheritance and polymorphism.

1.3 Assessing Students

As previously mentioned, programming is a hands-on discipline and requires considerable practice. It is more valuable to students if insightful comments can be made on the programs they have written. Therefore, a form of assessment is required, whether it be summative or formative. The structural nature of programming allows lecturers to split the teaching of programming into concepts, e.g. conditionals, meth-

ods, inheritance. These concepts can then be used as a basis for assessment.

Exams are the stalwart of the educational assessment arsenal. Programming gives you two main options when it comes to exams, a standard written exam or an online programming exam. The online exam is more suited to programming allowing students to both compile and test their solutions. It also allows students to write a program in an environment they should be confortable using. Another assesment method is via the use of Multiple Choice Questions (MCQs). MCQs let the lecturers cover several, if not all, details within a set conceptual area. A fine line exists, which when breached stop MCQs from testing knowledge and instead see how well students read the question or can learn obscure facts about the concepts being tested. Both exams and MCQs are expensive with regards to the time needed to organise and run, however, there is another option, coursework.

Regularly assessed coursework is a convenient way to measure students' progress. Not only can questions be written quickly, students do not need to complete the exercises in exam conditions. Course conveners can also benefit from the regularity of the exercises. Through the results, lecturers are able to see where mistakes are being made and whether the difficulties uncovered are general or concept related. However, every solution written needs to be marked. This is where problems are encountered.

1.3.1 Marking Exercises

In the wake of the "dot com boom", the numbers of students applying to study computer science rose sharply. Even now, several years after the apparent crash, student numbers are not insignificant. Whilst good for the Universities, it causes difficulties when attempting to mark programming solutions.

Traditional hand marking methods have their place in academia, unfortunately the marking of programming solutions is not one of them [AB99a]. Visually checking over a program to determine whether it is syntactically and dynamically correct, not to mention if it meets the set specification is a difficult and time consuming task. This is especially true when a large number of scripts are involved. In order to set regular programming exercises, there needs to be a fast turn over with regards to the marking of solutions. This routinely involves the lecturers enlisting help, usually from postgrad-

uates. This reduces the time needed to process all scripts, but introduces consistency issues. A form of automation was required to remove inconsistencies and improve the marking turn-around. Lecturers often took it upon themselves to write tools to speed the marking process along, automating such tasks as script collection. Over the years more tools were written, one such collection at the University of Nottingham eventually became Ceilidh [BBF+95, FHG96].

1.4 Existing Systems

Writing a program is not just a case of programming and then releasing the result, it is a cyclic process involving regular testing and refinement. The process can be assisted by useful feedback, however, the feedback needs to be provided on-demand to be of use. In order to analyse a program and provide on-demand feedback, the testing needs to automated. There are two distinct sets of programs that need to be considered, software designed solely for the testing of programs and education based software created with assessment in mind. The systems described below do not by any means consitute a comprehensive list, they are merely examples of the different types of system.

1.4.1 Testing Systems

The systems discussed in this section provide users with the ability to interact with their interface directly. There is no automation to allow users to create tests quickly and with very little effort, a high degree of programming competency is required to create the test initially. Once created, however, the test can be reused ad infinitum often with minimal changes required to reuse the test in different environments.

Java Robot Class

The Robot Class [Mic04] was created by Sun Microsystems for their 1.3 release of the Java programming language. It was designed for the purpose of automating tasks graphical user interfaces could perform, with a view to demonstrating the software created. The class allows the programmer to take control of all native input to

the operating system. This capability can be used to simulate a user attempting to operate the system under test and therfore could feasibly be adapted for testing purposes. Although the Robot has the potential to be extremely powerful, it's inflexibility is a major hinderance.

The robot needs to be told explicitly where each object in the interface is located. It does not handle changes made to the design of the interface without requiring a major rewrite to the robot.

In the hands of an experienced programmer, most dangers should be avoidable. However, because use of the robot involves control over the native input, it is possible to access the underlying operating system, be it accidently or with malicious intent, and cause great damage to the machine being used.

JUnit / JFCUnit

JUnit [JU] is a java framework designed for programmers to create test suites. These suites consist of testing sequences to be carried out to determine whether a program is working. JUnit was created for testing commandline driven programs, but has been extended and superceeded by JFCUnit which works with the Java Foundation Classes enabling the testing of grahical user interfaces.

The main advantage of the JFCUnit software is that with a few parameters, the test suites will attempt to locate the object required. With the test suites themselves being java programs, the opportunity for errors being present in the test suite exists. Also unless the tests have been previously written and can be reused, it can take a considerable length of time to create them initially.

GUITAR

GUITAR is a framework solution which integrates various tools and techniques to be used in the various phases of GUI testing [Mem01]. Of GUITAR's constituent components, the most interesting is that of the regression tester. Once the tests have been automatically generated and run, the regression tester checks the results supplied by the system for any invalid test responses. On subsequent testing iterations, any test that previously completed successfully is bypassed. The tests that failed to complete

returning an invalid result, most probably due to modifications to the GUI, undergo a repairing process. They theorise that by altering tests that previously did not complete, the major failings of the system will eventually be highlighted.

The power GUITAR has when testing graphical user interfaces are also it's hinderances when marking is also considered. For marking to be consistent across the board, every solution needs to be run against the same tests. By modifying the tests using the regression tester, the solutions will be subjected to different test sets.

1.4.2 Capture and Replay Tools

Of all the variations of testing software, this is by far the most commonplace. This is down to the fact that a large proportion of all commercial testing software falls into this category. Each piece of testing software may work in a slightly different way, but they are all based around the concept of the video recorder. They allow you to load your graphical user interface and set the system recording your actions. When later required users are able to replay the actions previously performed. Due to their commercial rather than educational value, this section has been curtailed with only one system being discussed.

qftestJUI

Of the many programs on the market for testing graphical user interfaces, one of the more advanced is qftestJUI [QFS], created by Quality First Software. qftestJUI works by allowing the users to record their test sets within the testing system. The software then splits the tests up into the separate actions performed, thus allowing the users to add and remove tests from specific places in the testing sequence.

The ability to alter the test sequence without having to rebuild it from scratch make qftestJUI a very powerful piece of software. However, it does not quite have the flexibility to deal with solutions from large numbers of users where the exact details of objects are not known. Precise information is required for the software to find the GUI objects.

1.4.3 Automated Submission and Assessment Tools

All the systems hereto mentioned were designed with testing in mind. There are systems which have been created with assessment in mind. The following systems have all been created to assist with the marking and teaching of programming.

Ceilidh

Ceilidh [FTHS99] was created at the Univeristy of Nottingham from a collection of unix shell scripts that had been developed to automate certain tasks. As the number of scripts increased it was decided they should be packaged up to provide not only the lecturer but also the students with the advantages they could provide.

Ceilidh's marking system was developed around the fact that it was able to mark programs from several perspectives. It had the capability to not only mark programs dynamically, i.e. does the program work providing responses consistent with the question specification. Ceilidh was able to perform marking directly upon the students submitted source code. These tests were concerned with the programming style and layout of the code.

In addition to the automated marking, Ceilidh was also the distribution system for not only the coursework exercises, but also the course notes. It was initially created to deliver a C course but it's extensible design meant that marking tools were rapidly developed for several other programming languages. This extensiblity contributed to Ceilidh's downfall, over the years it had effectively become a collection of extensions based around user suggestions received since it's inception.

Ceilidh was not without it's limitations. There was no network support, users had to log on to the server hosting the system to be able to use it. It also did not have a full X-Windows or PC-Windows graphical interface. Ceilidh was updated in 1998 when it became the more functional CourseMarker [FHTS00, FHST01] (formerly known as CourseMaster). CourseMarker, and especially it's marking subsystem will be discussed in detail in the next chapter.

BOSS

The BOSS system [LJ99, JL98], created at the University of Warwick, is an assis-

tive tool rather than a marking tool. It provides students with the information required to complete the task and also a means to submit their solution. BOSS's marking capabilities are somewhat limited, it is able to run command line driven programs against a set of test data but returns just a mark of either 100% or 0% with no feedback. Once all student submissions have been received, it randomly allocates all submissions to a list of designated markers for handmarking. Thus removing any bias from marking the scripts of known students.

CodeLab

CodeLab, formerly WebToTeach [AB99b], is a web-based interactive assessment environment. It's designers claim it ideal for introductory programming courses. It provides students with large numbers of questions of increasing difficulty on all topics covered. The answers which can range from single word answers to fragments of code are marked automatically by the system.

CodeLab only appears to mark either single line or short answer questions. Students using the system may understand the theory about the concepts the system examines, however, they will have no experience of writing complete programs themselves.

RoboProf

RoboProf [Dal99] is another web-based system. It is different from CodeLab in that it simulates an electronic course book. The students are provided with the course notes in HTML form and will not let them advance until they have mastered the chapter currently being studied. If the student answers a question incorrectly they are given another similar question to complete. This does promote learning, but cannot be used for assessment as the opportunity for inconsistencies is too great.

Chapter 2

CourseMarker

CourseMarker (CM) is a CBA system developed at the University of Nottingham to completely automate the assessment process for their first year programming module. It not only delivers the question to the student but also performs all marking and provides on-demand feedback to the students. It has now successfully been in use since 1998.

As previously mentioned, CourseMarker is a reincarnation of the Ceilidh system. Due to failings present in Ceilidh, CourseMarker was designed from scratch [FHH+01]. This allowed the designers to encorporate the requests for additional functionality that were not implementable under the old Ceilidh system. The redesign enabled the designers to encorporate knowledge about object-oriented methods, frameworks and design patterns. This provided major benefits in the areas of extensibility and maintainability.

This chapter is to outline the whole of the CourseMarker system before examining the marking subsystem in greater detail.

2.1 Overview

CourseMarker was initially designed with platform independence in mind. This was achieved via the use of the Java programming language. Java, being an interpreted language, run under the Java interpreter's Java Virtual Machine allows for the portability of programs between platforms. This removes the need for platform specific classes, reducing development time and simplifying technical support. Java offers the programmer an advanced exception mechanism and more importantly a way to dynamically link and unlink programs. Java also supports the development of distributed systems via the use of their Remote Method Invocation (RMI) mechanism.

2.1.1 Remote Method Invocation

Remote Method Invocation has allowed the logical processes of CourseMarker to be split into several separate subsystems. The advantage RMI has over the use of network sockets is that it effectively makes the network transparent to the programmer. CM's remote objects communicate with each other using a set of common interface,

which if required would enable the subsystems to be placed on separate servers.

2.1.2 Subsystems

After detailed analysis, Ceilidh's processes were extracted and seven subsystems were created. The responsibilities of each subsystem with regards to the operation of CourseMarker are both separate and distinct. These responsibilities are described below.

Figure 2.1: A high level view of CourseMarker's subsystems

Archiving Subsystem

The archiving subsystem is responsible for, once the submission and marking has been completed, collecting and storing a complete back up of the students solution. Not only are the student's files collected, but also a copy of the marking receipt file which stores the student's mark and shows where mistakes were made.

Audit Subsystem

The audit subsystem is solely for the use of the system administrators. By providing the other subsystems with the ability to maintain logs, it keeps track of what the system is doing at any point. This information is output to the server console so were something to go awry, the adminstrators would have a fair indication of what was happening prior to the malfunction.

Ceilidh Subsystem

The Ceilidh subsystem, named after the old Ceilidh system, handles the communication between the subsystems. It is responsible for establishing all the rmi connections between the subsystems. When clients try to connect, it is the Ceilidh subsystem that is queried to discern whether the other subsystems are up and running.

Course Subsystem

Each CourseMarker exercise consists of a collection of several different files. It is the responsibility of the course subsystem to provide access to these files when they are requested by the marking and submission subsystems. It also provides an object representation of all types of file that are encountered by the system, e.g. Java, C++ or text files.

Login Subsystem

The login subsystem attempts to recognise and authorise every attempted connection to the system. A list of users is created to assign students to modules. Passwords can either be individually assigned or CourseMarker can use their current system password. Only if the login process is successful is the client GUI loaded.

Marking Subsystem

The marking subsystem is responsible for the execution of the students' solutions against the sets of test data. The subsystem also calls the marking tools, collects the results and returns a mark along with feedback to the user.

The marking subsystem shall be discussed in greater detail later in this chapter along with what the marking tools are and how they function.

Submission Subsystem

The submission subsystem essentially controls the calls to the other subsystems. It is the submission subsystem that decides whether marking is to be performed or not. The submission subsystem not only checks the exercise parameters to see whether archiving or marking should occur, but it also checks the number of submission already made to see if the maximum has been reached or not. If marking is allowed then the subsystem makes the call and accepts the results which are then passed to the archiving subsystem for storage.

2.2 Marking Subsystem

Several desirable features were kept in mind during the creation of CourseMarker's Marking Subsystem. Such features included ensuring the security of CourseMarker, the ability to paramaterism the exercise settings, improved reliability and the most important from an assessment viewpoint, expressive feedback.

The view of the marking system differs depending on whether you are a student using CourseMarker or the course administrator. Whilst it is always important to consider the students viewpoint when creating courses and exercises, it is the internal details which are of interest to this research.

During the creation of an exercise, several files are used that can alter the way in which the system is to mark a solution. This enables the course administrator to have complete control over what is to be marked right down to the case of output text. The usage of the marking files will be described below, documentation does exist on how to create an exercise [Sym98].

2.2.1 Exercise Properties

It is possible to parameterise the exercise settings. The "`properties.txt`" file gives the course administrator the ability to alter the number of submissions open to students. In addition to this, it also allows them to turn off the automatic marking process, restrict the amount of output the student's program should have and also change the status of the exercise from open to late or even closed. If an exercise is set as late only students who have been given permission are allowed to submit.

2.2.2 Marking Tools

The actual marking processes performed by CourseMarker is controlled by the "`mark.java`" file. Within this file, the types of marking to be undertaken are defined. It contains calls to CM's marking tools and sets the mark distribution between the tools. Each call is passed not just the solution file, but also another mark file which contains the tests themselves. These tests, depending on the tool concerned, can contain anything from a search string to typographical test parameters.

CourseMarker currently has at its disposal, a collection of six different marking tools. Each tool runs a completely different set of metric tests against the students' solutions. These are essentially quality checks for the programs which then provide users with feedback on how their submission fared. These tools comprise of two java files. A control file, which collects all the files needed to perform the marking and sets up the system in preparation for the results. The second file is the actual tool itself. This file determines how the marking is to be performed, augmented by parameters provided in the marking files. The code contained within the java files is procedural enabling system administrators to write new tools with very little knowledge of CM's workings. The collection of tools currently available consists of the following:

Typograhical Tool

The typographical tool performs tests against the students source code, which analyse certain layout aspects of the program. There are several tests available [Gib95], the most commonly used of which check for correct program indentation, appropriate use of comments, amount of white space and line and indentifier length. The exact parameters used by the tests can be altered when creating the exercise.

Features Tool

The features test provides the capability to search through the submitted source code. This is of use, if for example the students have been asked to use certain methods or concepts in their solution to a problem. The functionality of the features tool is not just restricted to use on submitted source code, but can be used on any file. This allows for the analysis of files created during runtime. It also has uses regarding the security of the system as a whole.

Dynamic Tool

With the use of threads, this tool starts the students submission and supplies it with test data. Then with the use of the features tool, it checks the program output for the required responses. Unlike the other tools mentioned so far, it uses multiple files to assist in the marking, one containing solely test data and another containing the test

oracles. The oracles are comprised of sets of individual tests, each with their own mark value, search string and possible feedback options.

Flowchart Tool

In order to mark flowcharts, the tool converts the diagram into a BASIC program. Once the conversion is complete, it can be treated as a standard program submission and marked using the dynamic tool returning both marks and feedback to the student.

Object-Oriented Tool

The Object-Oriented tool checks program designs created by the students. It is able to test for completeness, correctness and accuracy. The relationships between components are also examined. Students can be penalised for the inclusion of classes that are not required.

CircuitSim Tool

The CircuitSim tool allows the students to model electrical circuits. The marking is performed by simulating the running of the circuit with students losing marks for incorrect wiring.

2.2.3 Feedback

Within the realm of assessment, whether formative or summative, feedback is equally as important as a well designed question. In his inaugural professorial lecture Hattie said "...the most powerful single moderator that enhances achievement is feedback" [Hat99]. Feedback enables lecturers to impart further information on the students. Correctly worded feeback can cause students to think critcally about mistakes they have made or reinforce a students knowledge with motivational comments. However, to be effective, feedback must be: constructive, timely and meaningful [Bon99].

The timing of feedback is important. If the delay between submission and receiving the feedback is too great, the work will no longer be fresh in the students mind and the comments will no longer have the same relevancy. CourseMarker has no problems on this count. Due to CM's automatic marking mechanism, it is able to provide

on-demand feedback. On-demand feedback is supplied as soon as the submission and marking process completes which is almost instantly.

Meaningless feedback has no use whatsoever. All comments returned in the feedback should be relevant to the tests being undertaken. CourseMarker has strict marking criteria meaning that at any point what the test is attempting to assertain is known explicitly. This allows the question designers to create feedback, which is closely related to the tests performed.

The final requirement of feedback is for it to be constructive. Whilst feedback needs to point out mistakes students have made, a comment of "Wrong!" is of no benefit. The majority of students would prefer the system to pinpoint the exact problem in the exact location that their work exhibits, but experience has shown that ultra-detailed feedback can be detrimental to the students learning experience. Therefore the feedback should indicate to the students, the direction the errors are taking their program in. CourseMarker's typographical tool is a good example of this sort of constructive feedback. The tool allows the question writer to supply five different feedback comments which are returned depending on how close to ideal a solution is.

CourseMarker delivers the feedback to students with the use of a "tree" GUI widget. Every CourseMarker marking tool returns feedback to the student that is assembled in this tree. The exact mark the student obtains is not shown, merely a coloured grade representation of the mark along with relevant feedback.

Figure 2.2: CourseMarker's Results Tree

2.2.4 Reliability

A major factor concerning systems for marking coursework is reliability. Students will rapidly lose faith and patience with a system that needs to be regularly restarted, especially when their marks are to hinge on it. CourseMarker's reliability comes from its capability to separate itself from each marking call. The extraction of each marking process means that were something to go wrong with the marking system or more likely with an incorrectly formed solution, it would not affect the system as a whole. The student would be awarded zero marks for that set of test data and the subsequent

set of test data would be run against the program as if nothing happened.

Other controls are in place to stop damage occuring to the system. These include timing the runtime of the students' solutions in addition to counting and limiting the number of lines of output the program being tested can produce. This all but removes the chance of programs containing infinite loops from interrupting CourseMarker's assessment process.

2.2.5 Security

The security of CourseMarker takes on two guises. The first and most obvious to the users is that of user authentication. The password protection mechanism is also encrypted to protect it from TCP/IP sniffing techniques. In addition the client is also checked for tampering via the use of a session key security feature.

The second part is more concerned with the protection of the system from malicious programs. Through the use of CM's own sandbox, the students are stopped from both reading and writing to files. Permission can be given to allow the students access to files with certain file extensions as required to set file reading/writing courseworks. As the students solution is being run by the system for testing purposes, the program will most likely have higher levels of privileges than when run by the students themselves. Therefore the programs need to be checked for any keywords that would indicate the code is malicious. CourseMarker compares the solutions to a comprehensive list of commands, the list includes such commands as delete, link and unlink to name but a few. If any of the list occur in the source, the marking process is instantly halted before the program is run and a security exception is thrown. This not only gives the student zero for their solution but also prevents any damage occuring to the system whilst taking a copy of the program to analyse what the student was attempting to do.

2.2.6 Extensibility

Careful design of the marking server has meant that CourseMarker is easily extensible. New marking tools can be created, or existing ones altered, easily without significant in-depth knowledge of how CourseMarker as a whole works. Due to the

tools being written procedurally, the only controls that CM places upon the tools are how the results are stored and returned to the system.

CourseMarker's internal state can be queried from within the marking system, therefore novel marking strategies can be authored that perform a variety of new tasks. For example, a marking strategy may be devised that compares the current student's submission with all the work that the student has ever submitted during the course, in order to establish whether the student is improving or not. In the latter case, CourseMarker would suggest to the student to seek advice and help from his/her tutor in order to solve any problems. Full details on CourseMarker's customisability can be found in a technical report [Sym01].

2.3 CourseMarker's Usage at Nottingham

As previously mentioned, CourseMarker has now been used in th University of Nottingham's Computer Science Department without problems since september 1999. The only major change to have affected the way in which programming was taught happened alongside the changeover to the CourseMarker system. The curriculum was changed so that instead of C, a procedural language, Java, an object oriented language, was to be taught.

2.3.1 Assessment

The students are subjected to continual assessment which takes a variety of forms. The majority of their assessment comes in the form of weekly exercises. The exercises are based on the topics they have been taught most recently and increase in difficulty each week. Not only are the students given exercises they must write from scratch they are also set exercises which involve debugging faulty programs. The bugs written into these programs are based upon common compilation errors the students have experienced in past years.

The students are also subjected to more formal assessment. Over the course of the year, they must sit several multiple choice tests, which cover everything from programming concepts to following through a few lines of code to discern whether they

can recognise what they correct output will be. The final formal assessment comes in the form of an online programming exam. This takes the form of a normal weekly exercise, but the test is performed on a secure network. The students must complete the exercise in the time available without the use of their notes or any assistance they could gain from the internet.

2.3.2 Course Coverage

The first year programming course at Nottingham is spread over the entire academic year. It is also split into halves, each being taught by different lecturers. The first half focuses on procedural programming and the fundamentals of java. It finishes off by touching on objects, but this is really the domain of the second semesters course. Alongside objects, they also cover other advanced topics as Graphical User Interface and concepts such as the Model Control View.

While the weekly exercises still encompass the topics covered in the semester two, the way they are marked changes. CourseMarker is only able to mark commandline driven programs. This means for the majority of the second semester the lab assistants are required to undertake the marking duties. However, for a student to be able to get real-time feedback on their solution, they must get it marked during the lab session. Therefore the marking scheme is significantly cut down compared to the exercises marked by CM.

This is the problem that has been tackled by this research. The rest of this thesis will attempt to break down the problem faced with regards to the automated marking of graphical user interfaces. In outlining an approach that can be used to solve the problem, this research will examine the tests required for sufficient test coverage along with the properties of graphical user interfaces that can assist the marking process.

Chapter 3

Graphical User Interfaces

Graphical User Interfaces (GUIs) were first introducted in 1973 by the Xerox corporation, but did not become ubiquitous until after such developments as the creation of the mouse pointing device. GUIs were designed to provide users with a simplistic way of interacting with the underlying computer system. To encourage faster acceptance of GUIs, the onscreen objects were given metaphorical names e.g. window, menu and document. Such was the take up of GUIs that now a large percentage of software released has it's own graphical interface. Not only have GUIs become more common place but they also account for a large proportion of work required to create a system. Surveys have suggested that GUIs account for over half the project time and also half the source code [Mye93].

3.1 What are Graphical User Interfaces?

GUIs are essentially no more than a facade used by programmers to mask all input and output processes. The removal of the command prompt and the limitation it imposed has allowed, through graphical representation, extra levels of complexity previously unobtainable. Examples of the added complexity include the ability to input / output multiple items of data simultaneously. However, advancements never come without a cost. Guis are no exception to this rule as they add to the list of considerations that need to be taken into account during the design of a system. This list includes the real time requirements of both the interface and program, and the need for added robustness due to the wider range of data that can now be entered by users.

The user interfaces themselves are constructed from a large number of objects. All the objects are customisable and allow the programmers to define where they appear on screen and what operational properties they and the system have.

3.2 Design and Construction

As previously mentioned, the design and creation of graphical user interfaces constitutes a significant proportion of a projects resources. This is in part down to their inherent complexity but also rests upon the fact that being an expert programmer does not make you an expert on interfaces or vice versa [Joh00]. It is a common occurance

Figure 3.1: Heirarchical view of GUIs

for an interface consultant to be brought in to assist programmers in the creation of new software.

3.3 Components

GUIs themselves consist of a large collection of objects, which allow users to define where objects appear on screen and what operational properties they and the system has. The objects are one of two different types: containers or components. The containers as their name suggests, are responsible for storing other objects and arranging them according to predefined layout criteria. Containers are not only able to store components but also other containers, which allows a degree of nesting within the interface and more control over the location of objects. The components are therefore the actual input / output tools of the system, such as text fields and buttons etc. They provide the program with all its functionality.

what how why

3.4 Heirarchies

As you would expect from a object oriented langauge, graphical user interfaces are very heirarchical in their design. So much so that they provide us with the perfect opportunity to experience object oriented programming. The hierarchical structure exhibited by GUIs is a testament to how inheritance and other OO techniques can be put to good use whilst giving us intrinsic information about what we can expect to find at any point. Due to the OO nature we are able to think of GUIs as tree structures, at the very top node of this tree we have the window construct. Following it down we encounter the containers down to the roots where we find all the components. We know these to be either components or empty containers, as they do not have any internal objects.

structure of guis -¿ building up to descent parser

3.5 States

what are they transition between

Bibliography

[AB99a] D. Arnow and O. Barshay. On-line programming examinations using web to teach. *Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education*, pages 21–24, June 27 - 30 1999.

[AB99b] D. Arnow and O. Barshay. Webtoteach: An interactive focused programming exercise system. *ASEE/IEEE Frontiers in Education Conference*, November 1999.

[AT95] G. Åkerlind and C. Trevitt. Enhancing learning through technology: When students resist the change. *ASCILITE 95 - Learning with Technology*, December 3-7 1995.

[BBF+95] S. Benford, E. Burke, E. Foxley, N. Gutteridge, and A. Mohd Zin. The design document for ceilidh version 2. Ltr report, Computer Science Department, The University of Nottingham, 1995.

[Bon99] A. Bone. *Guidance Notes: Ensuring Successful Assessment*. Warwick Printing Company Limited, 1999.

[CE98a] D. Charman and A. Elmes. Computer based assessment: A guide to good practice. *Volume 2*, 1998.

[CE98b] D. Charman and A. Elmes. Computer based assessment: A guide to good practice. *Volume I*, 1998.

[Dal99] C. Daly. Roboprof and an introductory computer programming course. *Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and Technology in Computer Science Education*, pages 155–158, June 27-30 1999.

[dJ05] P. de Jager. Teaching old dogs new tricks. *Computerworld Canada*, January 24 2005.

[FHG96] E. Foxley, C. Higgins, and C. Gibbon. The ceilidh system : A general overview. Ltr report, Computer Science Department, The University of Nottingham, 1996.

[FHH+01] E. Foxley, C. Higgins, T. Hegazy, P. Symeonidis, and A. Tsintsifas. The coursemaster cba system: Improvements over ceilidh. *Fifth International Computer Assisted Assessment Conference*, pages 189–201, July 2-4 2001.

[FHST01] E. Foxley, C. Higgins, P. Symeonidis, and A. Tsintsifas. The coursemaster automated assessment system - a next generation ceilidh. *Workshop on Computer Assisted Assessment to support the ICS disciplines*, April 5-6 2001.

[FHTS00] E. Foxley, C. Higgins, A. Tsintsifas, and P. Symeonidis. The ceilidh-coursemaster system, an introduction. *4th Java in the Curriculum Conference*, Jan 2000.

[FTHS99] E. Foxley, A. Tsintsifas, C. Higgins, and P. Symeonidis. Ceilidh, a system for the automatic evaluation of students programming work. *CBLISS 99*, July 2-7 1999.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley Professional, first edition, 1995.

[Gib95] C. Gibbon. Writing typographically sound programs with ceilidh. Technical report, The University of Nottingham, 1995.

[Hat99] J. Hattie. Influences on student learning. Inaugural professorial lecture, University of Auckland, 1999.

[JL98] M.S. Joy and M. Luck. The boss system for on-line submission and assessment. *Monitor: Journal of the CTI Centre for Computing*, pages 27–29, 1998.

[Joh00] J. Johnson. *GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers*. Morgan Kaufmann, 2000.

[JU] Junit testing framework documentation available at http://www.junit.org.

[LJ99] M. Luck and M.S. Joy. A secure on-line submission system. *Software - Practice and Experience*, pages 721–740, 1999.

[Mem01] A.M. Memon. *A Comprehensive Framework for Testing Graphical User Interfaces*. Ph.d. thesis, Faculty of Arts and Sciences, University of Pittsburgh, 2001.

[Mic04] Sun Microsystems. Java robot class documentation is available at http://java.sun.com/j2se/1.5.0/docs/api/java/awt/robot.html, 2004.

[Mic05] Sun Microsystems. Java programming language specification. Technical report, 2005.

[MMM04] M. McCulloch, H. Macleod, and N. Mogey. Assessing online - an overview. *Reflections on Assessment*, 2, 2004.

[Mye93] B.A. Myers. Why are human-computer interfaces difficult to implement? Technical Report CMU-CS-93-183, Carnegie Mellon University, July 1993.

[QFS] qftestui, the java gui testtool.

[Str00] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, third edition, 2000.

[Sym98] P. Symeonidis. Creating an exercise using coursemarker (formerly javaceilidh). Technical report, LTR Group, 1998.

[Sym01] P. Symeonidis. An in-depth review of coursemaster marking subsystem. Technical report, LTR Group, 2001.