A System to Mark Programs Automatically

Ken Ngo-Pham, John Bigham, Julian Rodaway, Laurissa Tokarchuk

Department of Electronic Engineering, Queen Mary University of London, London, UK
john.bigham@elec.qmul.ac.uk

Abstract

This paper describes a system to automatically mark student programs. It is a real time examination system that provides an environment for the compilation and execution of software and then marks it according to criteria and solutions provided by the examiner. It has been designed as a client server system supporting many simultaneous clients, with a modular structure that provides flexibility, allowing components to be added for new courses and kinds of questions without having to change the existing core structure. It is hoped that by sharing this software with others, a set of modules can be constructed to examine a range of programming questions. This report outlines the structure of the system. It is currently being evaluated in class assessments for software courses in network programming.

1. Introduction

With increasing class sizes there is more pressure to find effective ways of teaching software skills within the resources provided. In response to this there has been a rise in the use of multiple choice tests to assess software skills and this is now common in universities and in the accreditations of Microsoft and Sun. However, it is arguable that such tests do not test the ability to create a segment of software that can run. Additionally, students who fail software courses are often those who have not practiced enough. Automatic assessment can support such students with graduated exercise, and be used as a way of ensuring that the students are indeed doing the exercises and not falling behind. It is not claimed that the tool described here is a substitute for expert advice but it can provide substantial assistance.

There has been a long history of tools to support programming assessment, particularly for first programming courses, e.g. University of Bristol, U.K [1]. Many such systems are static, in the sense that the student submits the source and then it is checked by the marking system offline. However, CourseMarker [2], which is one of the best known systems, has a client server architecture. CourseMarker was developed from the previous Ceilidh system. Ceilidh has been evaluated (e.g. [3]) and shown to be successful, significantly reducing the burden of marking on the lecturer while not adversely affecting the student.

Just as in CourseMarker, we want the students to program exercises as they would normally do in the laboratory, and get a mark immediately, but other objectives differ. We want to provide support for a variety of quite different software courses and to allow users of the software to contribute modules geared to the software and kind of assessments they want to make. We have been testing the tool on network programming applications, applications using XML, and HTML pages, as well as introductory Java courses, and such examples are illustrated. Other tests are possible. The motivation comes from the recognition that software courses change all the time along with changes in technology, with many software courses now teaching e.g. TCP/IP, client server architectures, web service architectures. A flexible open source tool should allow users to specialize it to their needs with modest effort. We want to encourage contributions from lecturers around the world, so that a repertoire of different questions for different applications is available.

This paper describes some design aspects and the implementation details behind the system; the process of writing exam questions; the process of conducting an examination; and the process of marking of the examination. It also illustrates the application to a diverse set of marking tasks and discusses the structure that allows the system to be loosely coupled.

Proceedings of the Fifth International Conference on Quality Software (QSIC'05)
1550-6002/05 $20.00 © 2005 IEEE

2. General Overview of the System

Figure 1: The three interlinking parts of the system.

This exam system consists of three key parts as indicated in figure 1:

- The QCA (Question Creation and Administration) component represents the desktop application used by lecturers to create questions and marking schemes. The entities created are called exam components.
- The EERTA (Examination Environment and Real Time Assessment) component is a web application that provides the web based examination and marking functionality.
- The exam components are used by both the desktop application and the web application. The QCA uses the exam components to create persistent serialized data in XML format (indicated as specifications in figure 2) that are then de-serialized by the EERTA and transformed into the required objects. This is illustrated in figure 2.

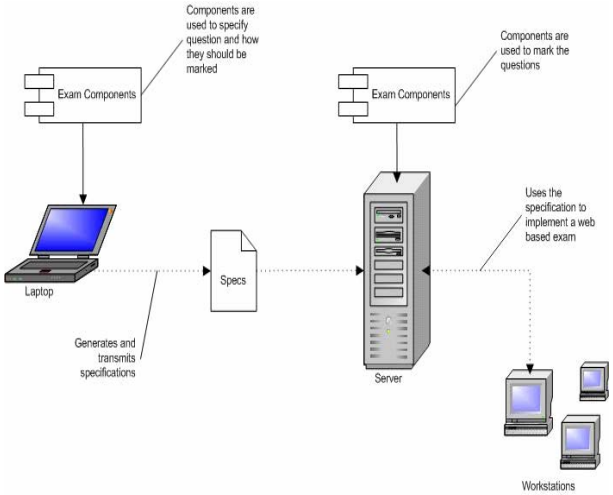

Figure 2: Overview of the system in operation

The process of writing examination questions, conducting an examination and the marking are modelled by the system through the implementation of two main parts, viz. the QCA and the EERTA. The QCA is used by lecturers to generate implementations of questions and implementations of various criteria to mark the question through the creation of the exam components. These components are then passed onto the EERTA – the web application, and are used to create a web based examination and the marking according to the specification. The exam components form a library of code used to model the exams, the questions and the marking criteria. The importance of the exam components is that they are independent of any front end. They allow a GUI (of which the QCA is an example) or web based interface to control and manipulate the library elements. The questions, marking criteria and answers are not hard coded into the system. This allows users to create new ones, change existing or reuse existing ones without modifying any pre-existing code used by the system. The flexibility behind the system allows contributors to add their own questions and criteria which can be then be reused by others.

Every question is represented in the exam components as a Question object. The Question class contains *accessors* and *mutators* for the necessary attributes, such as the text of the question and the total mark allocated to a question. It also contains one or more marking schemes. Each marking scheme consists of a set of marking criteria corresponding to the facts that the answer should contain. To allow flexibility it is possible for lecturers to offer alternative marking schemes for a single question, as there could be multiple approaches to the question. The system will mark using all alternative marking schemes of a single question and award the highest awarded mark.

The criteria used for marking varies quite substantially across different subjects and are likely to change to match evolving curricula. This changing factor has been incorporated into the design by allowing lecturers to write their own criteria depending on the needs of their students. Each criterion is implemented using a small unit of code that tests the student's answer code on whether or not it satisfies the encoded criterion. Each criterion is an implementation of the Test interface. Our derived Test API uses a Test interface (figure 3) to provide the flexibility needed to handle the different context each test requires in order for it to match each criterion. The Test interface also provides the uniform method signature needed by the system to process each Test sequentially. The methods are described more fully in Table 1. Each test is atomic in that it only tests for a single property which can either be present or absent in the code. Tests are independent of each other to allow a collection of them to be processed sequentially. They allow answer code to be examined and executed without any type of

modification. This maintains the integrity of other tests and the answer code.

3. Examples

This section looks at two examples of tests, which gives an idea of the capabilities of the system.

3.1 Java Bean Example

The following example illustrates how the tests derived from the Test interface can be used to mark a question. The exam question from figure 4 asks the student to write a Java bean with specific methods and variable types. To understand how a Java bean question can be marked it is important to understand that a Java bean is similar to any other Java class except it is written to a specific structure.

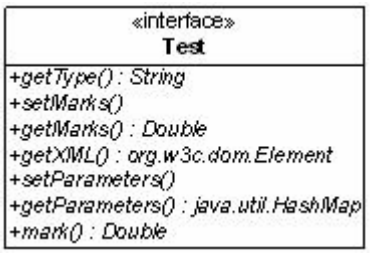

Figure 3: The Test interface

Method	*Purpose*
getType	Returns the name of this test in String format
setMarks	Mutator to set the number of marks awarded for this test
getMarks	Accessor to the number of marks awarded for this test
getXML	Converts the test into an XML element for serialization
setParameters	Mutator to set all the varying parameters (attributes) needed for this test
getParameters	Accessor to all the parameters (attributes) used in this test
mark	Performs the test and return 0 (fail) or the specified full marks (pass)

Table 1: Purpose of the methods in the test interface

Every Java bean must:
- implement the java.io.Serializable interface
- be well encapsulated so that all instance variables are private and methods are public
- provide accessor and mutator methods for each of the instance variables with method names relating to the variable names according to well defined rules.
- have a no-argument constructor

> A user's details consist of the name of the user held as a String, the email address of the user held as a String and the age of the user held as in int.
> Write a bean for the user details called UserData that includes all the methods assumed for a bean in a JSP. Also include a method toString() that converts an instance of UserData into a meaningful string. Any format is acceptable. This will be used later to help output instances of UserData.
>
> **[5 marks]**

Figure 4: A part of a network programming course exam question 2003. (The latter part involved the creation of JSPs to use the bean developed).

	Marking Scheme 1	*Marking Scheme 2*
Criterion 1	(1 mark) public no argument constructor	(1 mark) accessor/mutator for "user"
Criterion 2	(1 mark) encapsulated variables	(1 mark) accessor/mutator for "email address"
Criterion 3	(1 mark) implement the Serializable interface	(1 mark) accessor/mutator for "age"
Criterion 4	(1 mark) accessors/mutators for one instance variable	(1 mark) toString method
Criterion 5	(1 mark) toString method	

Table 2: Two alternative marking schemes for the same question

Assuming that the five marks for the question in figure 4 are for each of the four properties of a Java bean listed

above plus one for writing the method "toString", the lecturer may award discretionary marks to some students who score only a few marks but have written something else of value. This can be modelled with the exam components by defining two or more marking schemes for this question as shown in table 2.

In table 2, if the student writes a class which encapsulates the instance variables and provides three pairs of *accessors* and *mutators* along with the 'toString' method and nothing more, then marking scheme 2 will return the maximum number of marks of scheme 2 – 4 marks. In order for the student to score full marks for this question, they are required to write code to fulfil the marking scheme 1. It is important that the lecturer plans out clearly how they would like a question to be marked and to also consider alternative marking options. Because of the discrete nature of the system, any unplanned answers that can be considered correct will return zero unless defined. The lecturer has the option to define as many marking schemes they feel appropriate, in order to offer a more flexible marking option to cater for the wider range of students.

3.2 Deployment descriptor for Tomcat

The deployment descriptor in Tomcat is a XML file used to define which *URL requests* a Servlet will respond to within a particular web application. Additionally, the deployment descriptor has the option to allow users to implement a security policy to constrain resources within the web application. The Tomcat deployment descriptor opens the opportunity for students to learn and apply their web and security knowledge through the implementation of a security policy using XML syntax. How XML tags are used to specify security constraints is summarized in table 3.

If a student were asked to answer the question in figure 5, we can assume that a possible marking scheme is to award 1 mark for each of the five points from table 3. The question is asking the student to specify in the deployment descriptor the constrained URL, the constrained HTTP method, the authorized role, the authentication method and the level of confidentiality. Since there are no alternative tags to implement, a single marking scheme suffices. However it is possible that when students do write the code, they may misspell or omit specific tags and the lecturer can accommodate this by specifying alternative marking schemes that can search for alternative properties. Table 4 lists the criteria for the question in figure 5.

> Write the deployment descriptor for Tomcat to only allow access through the HTTP method 'GET' to the URL "/member/documents/" so that only the role 'members' can have access. Make sure users are authenticated using standard BASIC encoding and the requested URL is delivered confidentially.
> **[5 marks]**

Figure 5: Possible Tomcat deployment descriptor question.

It is clear from table 4 that there are several properties that marks can be awarded for. Although security constraint is defined using specific XML tags, marks can be also be awarded for the structure of the XML document which is just as important as the mark-up value of the tags. Generally, the exam system offers the flexibility of marking and its utilization can depend on the goals of the lecturer.

XML	Purpose	Security Type
<url-pattern> /aboutus/secret/ </url-pattern>	To state the URL resource is constrained	Constrained
<http-method>GET </http-method>	To state the HTTP method to access a resource is constrained	Constrained
<auth-constraint> <role-name>Admin</role-name></auth-constraint>	To state that only a particular role has authorisation and users must be authenticated	Authorisation
<login-config> <auth-method>BASIC</auth-method></login-config>	To state which mechanism is used for authentication	Authentication
<transport-guarantee> CONFIDENTIAL </transport-guarantee>	To state that the request resource should be transmitted with confidentiality	Confidentiality

Table 3: Sample XML tags used to display specific security constraints.

	Marking scheme 1	Marking scheme 2
Criterion 1	(1 mark) '/member/documents/' is marked up by tag <url-pattern>	(1 mark) tag <role-name> is child of tag <auth-constraint>
Criterion 2	(1 mark) 'GET' is marked up by tag <http-method>	(1 mark) tag <auth-method> is child of tag <login-config>
Criterion 3	(1 mark) 'members' is marked up by tag <role-name>	(1 mark) tag <role-name> is child of tag <auth-constraint>
Criterion 4	(1 mark) 'BASIC' is marked up by tag <auth-method>	(1 mark) 'BASIC' is marked up by tag <auth-method>
Criterion 5	(1 mark) 'CONFIDENTIAL' is marked up by tag <transport-guarantee>	(1 mark) 'CONFIDENTIAL' is marked up by tag <transport-guarantee>

Table 4: Lists two possible marking schemes to mark the question.

4. Marking Java Code

The technique used to mark questions is not complicated. One approach would involve the process of coding a compiler that would take a source file and examine it line by line testing for the criteria defined. Although this is a possibility, the performance of such a technique would require high level hardware resources when marking hundreds of student's source code. The difficulty of writing complex code in the 'mark' method could well discourage many lecturers from writing their own tests.

When a program source is compiled, information about the structure of the code is lost as lower level code is produced. However, Java preserves the structure information as metadata with the generated code. Programmers can access this metadata through a package known as the reflection package. The reflection package allows programmers to access information such as the methods, the super classes, the variables of an object at runtime. This package was designed to allow vendors to create powerful and rich software development environments to aid programmers. An example used in such systems is to underline or highlight undeclared variables written in these software development environments, or to list as a table all the methods a currently viewed segment of source code contains, on a side panel. This unique property offered by Java is the underlying mechanism we use to allow Java source code to be marked.

To demonstrate how the criterion of 'implements the Serializable interface' is implemented through the use of java.reflection package, figure 6 demonstrates how the 'mark' method works.

```
public class IsSerializable implements
Test{
…
public double mark(Object object) {
  Class c = object.getClass();
  Class[] theInterfaces = c.getInterfaces();
   for(int i=0;i<theInterfaces.length;i++){
            if(theInterfaces[i].getName().equa
ls("java.io.Serializable"))
     return marks;
}
return 0.0;
}
…
}
```

Figure 6: IsSerializable test mark method implementation. Note that the variable marks is defined outside the method mark()

In figure 6, the 'mark' method has an argument of type 'Object.' This object will be the students answer code compiled and instantiated. The 'mark' method then tests the object to find out whether or not this object implements the Serializable interface. If the object contains the properties defined by the method then the marks are awarded, otherwise nothing is awarded. The use of the reflection package here allows us to check for specific properties at runtime of any object; the only requirement is that the student's answer code becomes instantiated before marking. This test can be re-used for many different questions and is not bound to only Java bean type questions. It is hoped that by using the reflection package in this way, lecturers can code very simple discrete tests that can be re-used by many for any given situation.

5. 5 Summary and Future Developments

In the previous sections we have attempted to describe the structure and capabilities of the marking system. There are however, limitations. Key limitations of the approach adopted are:

The approach to testing marks the program by executing the students program in the context of data and correct code provided by the lecturer. If the program

compiles correctly then the checks look for certain patterns in the code, and make comparisons with e.g. string or numerical output. There is no proof that the program is logically correct or otherwise.

No account is taken of style. In fact some measures of this are possible, but this has not been the focus of the work.

However, many simple exercises can be tested. Laboratory assessments are reasonably formulaic and appear well suited to assessment by the scheme used. The goal is not to have a system to mark large coursework, but to give help with smallish structured questions, very similar to end of year examination questions.

At present the system is being used to tests several groups of students in topics such as XML applications, HTML and many aspects of Java including Java beans, Servlets and core Java programming. Further 'tests' currently in consideration include JSPs, extending the XML to cover RDF and SOAP, web services, in particular WSDL files and programming using the Java API for XML remote procedure calls. This system can develop the skills of students by allowing the opportunity to practice their programming skills in a monitored way. Lecturers can teach concepts in lectures and then use the system to help the student apply these concepts in a program. How well students apply these concepts can help lecturers develop alternative strategies to teach them.

Looking ahead, it is possible with the help of contributors, to turn the system architecture into a service oriented architecture whereby a central registry is used to list all the different tests from various contributors, the purpose of each test, the parameters required, and to be able to download the class file for local simulation. During the marking process, the registry can be used to locate which computer has the specific test deployed and how that computer wants it's request structured as a mechanism to distribute processing across the network. Further to this, if hopefully an increase of resources becomes available, the system can be extended to cover other languages such as C++, C#, Perl or Visual Basic which without doubt is currently possible.

Having the examination and marking process automated electronically will not only free more time for the lecturer but can also provide the opportunity to capture data that can then be processed and analysed efficiently. Reports can be generated after exams which show statistics that could for example, list the questions that students spend the most time on, which students failed to complete which questions, the question(s) that the majority of students score badly in, the typical compilation errors students made and in which questions students do well in. The statistics provided by such a report can help the lecturer to understand the weaknesses and strengths of students and allow the course to be taught more flexibly where the whole class is monitored and their progress moves forward as a whole.

6. References

[1] http://www. cs.bris.ac.uk/software/mark/index.html

[2] University of Nottingham U.K. http://www.cs.nott.ac.uk/CourseMarker/

[3] S. P. Foubister, G. J. Michaelson & N. Tomes Learning Technology Centre, Department of Computing & Electrical Engineering, Heriot Watt University "Automatic assessment of elementary Standard ML programs using Ceilidh", Journal of Computer Assisted Learning, Volume 13 Issue 2 Page 99.

[4] Ingrid Marson (2005, April). Programming apprentice scheme demanded. Retrieved April 22 2005, from UK Builder's website: http://uk.builder.com/manage/work/

[5] W3C HTML Home (modified 2005, March), Retrieved March 29 2005, from W3C's web site: http://www.w3.org/MarkUp/

[6] Freeman & Freeman (2004). Keeping your objects in the know. Head First Design Patterns (pp. 51). Sebastopol, CA: O'Reilly (0596007124)

[7] Freeman & Freeman (2004). Well Managed Collections. Head First Design.