JavaMarker Extended: An Eclipse Plugin to Mark Java GUI Programs

Marzieh Ahmadzadeh
Dept. of Comp. Eng. & IT
Shiraz Univ. of Technology
Shiraz, Iran

Mahsa Janghorban[1]
Dept. of Comp. Eng. & IT
Shiraz Univ. of Technology
Shiraz, Iran

Behnaz Jamasb[1]
Dept. of Comp. Eng. & IT
Shiraz Univ. of Technology
Shiraz, Iran

ABSTRACT
Graphical user interfaces are an important part of today's application, which requires academic staff to teach the principles and to ask students to deliver assignments on designing those interfaces. The considerable number of students in this filed makes the process of marking difficult and time consuming. Therefore the existence of an automated marking system seems inevitable.

Graphical user interfaces however, for some reasons are difficult to be marked automatically. Amongst these difficult issues are the event driven nature of graphical user interfaces (GUIs), existence of several components in Java that do the same job and can be used interchangeably and the possibility of designing a program in a complete different ways.

This paper elaborates these difficulties and introduces an Eclipse plugin, which is an extension to previously introduced system, designed to mark GUI programs.

General Terms
GUI Marking, GUI Grading, Automated Assessment, Computer

Based Assessment, Java

Keywords
Eclipse, Plugin

1. INTRODUCTION
Graphical user interfaces have spanned more than three decades started from Xerox product [8] in 1977 and evolved to be a major part of each program. As a survey in 1993 [12] reported, around 50% of development time is spent on GUI. The popularity of graphical interface is due to a friendlier look and better interaction that it provides in comparison with text-based systems. For this almost all of today's applications are equipped with a graphical interface.

This encourages universities to undergo major changes and to teach the design of graphical user interfaces. The major issue in teaching GUI is the difficult task of marking students' assignments. Hiring assistants to do the job may lead to inconsistence marking. To overcome this labor work, the development of an automated marking system seems unavoidable.

Although a couple of marking systems have been introduced in the literature [1, 2 ,3 ,4, 5, 9, 10, 13], a majority of these systems have been designed to mark text-based programs. Only a few of them have focused on marking graphical interfaces [6,7,14]. It seems that none of these systems have been developed to work in the predefined settings of the labs (i.e. they have been implemented as an independent system). Since the editor for programming course in Shiraz University of Technology had been chosen to be Eclipse editor [15] we decided to implement a plugin for Eclipse to mark students' assignment. This plugin which is called JavaMarker is an extension to previously implemented plugin, which was responsible for marking text-based programming assignment written in Java [1, 2].

Developing graphical interfaces is different from that of text-based programs and additional issues need to be taken care of. These issues will be covered in section two of this paper. In next section the developed plugin will be explained in detail. For this the architecture of the system, the format of test cases file, the views that is generated by the plugin and some snapshots of the executed marking system will be brought. In fourth section a review of the literature will be given. And finally the work that will be carried out in the future will be discussed in section five.

2. THE PROBLEM OF GUI
To efficiently interact with a computer program, Graphical User Interfaces were introduced. A number of so-called components such as frame, panel, button, etc., which interact with each other make a GUI. In such kinds of programs all the inputs and outputs is done via those component. For example you enter your information via a text field, a slider, a check box button etc. and you view the output in a panel, label or something similar. This is the sole difference between a command-based (text-based) program and a program with graphical user interface.

Although the difference between command-based programs and graphical user interfaces is the way in which data is entered and outputted, the marking of such programs is not that easy and includes several complexities.

One of the problems that a developer of an automated system faces is the variety of the ways that a programmer can choose to implement the given problem. For example if your program is about to get user date of birth, you have the choice of implementing this by a text field or a dropdown list or a combination of a text field and radio button and so on. This leaves the developer of such an automated system with alternatives that in most of the cases are not predictable by grader and this makes the marking even more difficult. What we

1- Both the co-authors have had the same contributions.

meant here by alternatives is the variety of possible layouts that a programmer can design to solve the given problem. Moreover, each component can operate differently which adds to complexity of marking. Thus it is unlikely that any two submitted program will be similar.

Another complexity of developing such a system is the properties that each component can have including size, location they appear, color and so on, which together form the aesthetic design of the program. Further, when a program runs, its components can change their states which provide more options for a marking system to test.

Further, event-driven nature of graphical user interfaces allow users to click anywhere in the created screen, which means there are many possible ways that input can occur. In other words the large number of available options for a program to receive an event makes the testing of the program even more complicated.

All these complexities together do not allow a grader to statically define input and test output against the entered input. In other words a grader should look inside the code and check whether each component does the proper job, which is responsible for. Also for the same reason a strict program specification should be provided to actually limit a programmer to use any available Java feature. For example if a text should be entered and the use of *textfield* is suggested in program specification, a programmer must follow this and will not be able to use *textArea* for instance.

Therefore one way to approach marking is to consider each object (i.e. component) and its state one at a time and compare it with expected state and award mark accordingly.

3. JavaMarker

JavaMarker as an Elcipse plugin was first designed to mark command-based programs [1, 2] and then extended to mark graphical programs. Although the latter is in early stage of its life, still it can be used to mark students' graphical programs. At the moment the icon representing this plugin is separate from its ascendant's [1, 2] icon since the process in which the marking is taken place is different from each other. However the process in which the right assignment is chosen to mark is similar to what is explained in [1, 2]. Therefore for the time being these two plugins are shown with two different icons and will soon be integrated to one.

3.1 Eclipse Plugin Architecture

As can be seen from figure 1 JavaMarker consists of several parts. First, students submit their program, which will be held in a repository. Then the plugin uses test cases file, which has been provided by a system administrator (i.e. a teaching assistant) and students' program from repository and marks the program. A report is issued for the student which can be seen in a view in Eclipse and another for lecturer in another repository.

What actually happens in Javamarker plugin is that it scans the Java program that has been submitted and removes nonessential lines of code (i.e. comments) and separates each declared object (component) and its features in some vectors. Features such as size, color, etc., which relates to aesthetic design, has not been considered in this stage. The information that are collected for an object is features such as the right definition of object itself, definition of any kinds of *listeners* on that object and to which container it has been added. On the other hand, the information, which resides in the test case file, is fetched and put in another vector. Then the comparison is taken place and mark is awarded accordingly.

For example, a program is supposed to define a *JButton*, add it to a *JPanel*, be registered with *ActionListener* and exit the program if the button is clicked. All of these definitions and operators are retrieved from both the actual program and test cases and then compared.

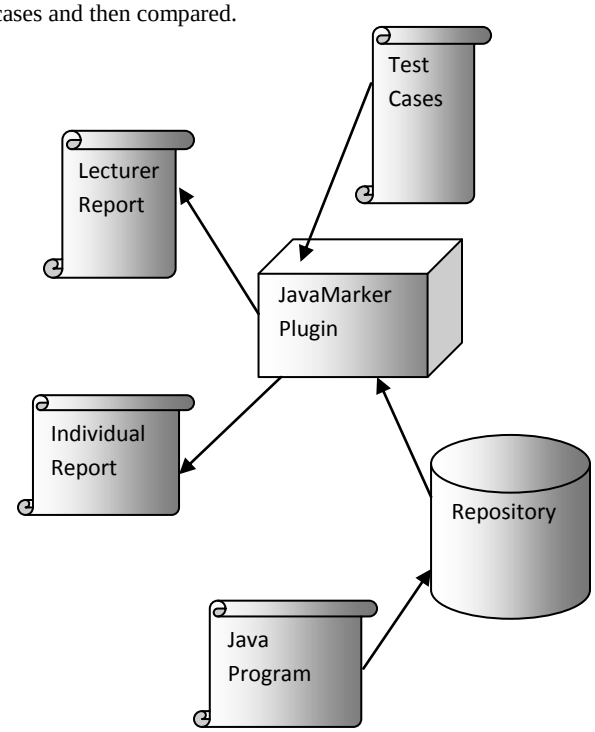

Figure 1: GUI Marker (JavaMarker) Architecture

One important task that is done by JavMarker is to recognize all the objects that work together. For instance a couple of buttons and textfields might together enter some information in a file and a couple of checkboxes and buttons print a report on screen. These sets are distinguished and their interaction is tested.

JavaMarker has been designed to mark the programs that are syntactically correct. Therefore if any compilation error exists none of the process explained above will be run and no mark is awarded.

3.2 Test Case Format

As said before the components (i.e. objects) that works together are distinguished. This helps to create an effective test case. As can be seen from Figure 2, which is a sample graphical user interface, the components that have been defined are a radio button, three checkboxes, two text fields, three buttons and one label. The first set (i.e. radio button, a text field and a button) form one group that works together. User chooses one of the options of radio button, enter a text in the text field and click the button to print the entered information into a file. The Second set, which comprises of three checkbox and a text field and a

button also form another working group. Clicking on the button leads to printing information of this group to a file. The third group simply consists of one button that exit from the program.

Figure 2: A sample graphical user interface

Therefore the test case file should look similar to Figure 3. The numbers (i.e. 1 , 2, 3) separates the group of components. The other group of numbers (i.e. 20, 50, 30) shows the percentage of the marks that will be awarded if the corresponding group of components work correctly. At the bottom lines of each numbered line, the components that form a group are introduced. The order of objects that comprise a component is not important and is recognizable by our system.

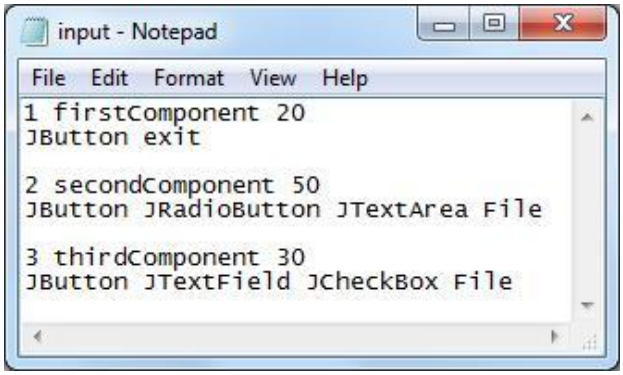

Figure 3: Test cases corresponding to Figure 2

3.3 Plugin View & Icon

Figure 4 shows a view of Eclipse along with our installed plugin. It can be seen that this plugin has created a view, which is beneath the editor area. This view is where the feedbacks generated by our marker are shown to students. For each program depending on the level of difficulty it has, a few submissions are allowed. If, for instance, two submissions are possible, after first submission students will see the comments in the result view and consequently can improve the program based on the given comments

In upper right part of Eclipse is an icon, created by our marker, in order to allow students to submit their program. Clicking on this icon shows a dialog box, in which student chosen the assignment they would like to submit.

4. ASSESSMENT OF PROGRAMMING ASSIGNMENT

Programming module is an important module that not only is taught for Computer Science but also for several other engineering majors. Success in this module is dependant to how effectively students practice. Therefore one way to make sure that enough practice has done by students is to give them weekly homeworks. With a large number of students enrolled for the module, teaching staff face a huge number of homework to mark. There are of course a few teaching assistants available to help but this leads to inconsistency in marking. To overcome this problem automated marking systems are developed. Although these systems are not able to be as intelligent and accurate as a human in the process of marking but they offer some benefits in several ways.

First, the marking will be consistent for all of the homeworks and throughout the semester. Second, students will receive an instant feedback on the program they have written. This is invaluable in terms of learning. That is why in most automated systems, students are allowed to submit their program more than once. This gives them the opportunity to think about their mistake and learn from it. Third, teaching staff will be able to enforce students to practice in the framework that they set if required. Finally, the process of marking will take considerably less time.

In this part we will describe the tools that were designed to mark text-based programs and graphical user interface separately in two following subsections.

4.1 Text-Based Assessment Tools

Development of marking text-based systems have spanned nearly two decades [1, 2, 3, 4, 5, 8, 9, 12]. These systems evaluate programs written in C or Java and check the correctness of the program and the quality of the code in terms of style. However they have chosen different approach to marking. For example BOSS system [9] provides all the information that one needs to complete the task of programming but does not issue any mark. Marking is done manually in this system.

Coursemarker [10] not only mark students' program based on what is given in program specification, but also provide an environment in which students receive program specification, develop their programs, submit them and receive the corresponding mark.

Amongst the systems that were introduced, only JavaMarker [1, 2] works in the same environment that students would normally develop their code (i.e. Eclipse). This helps students to focus on the task of programming.

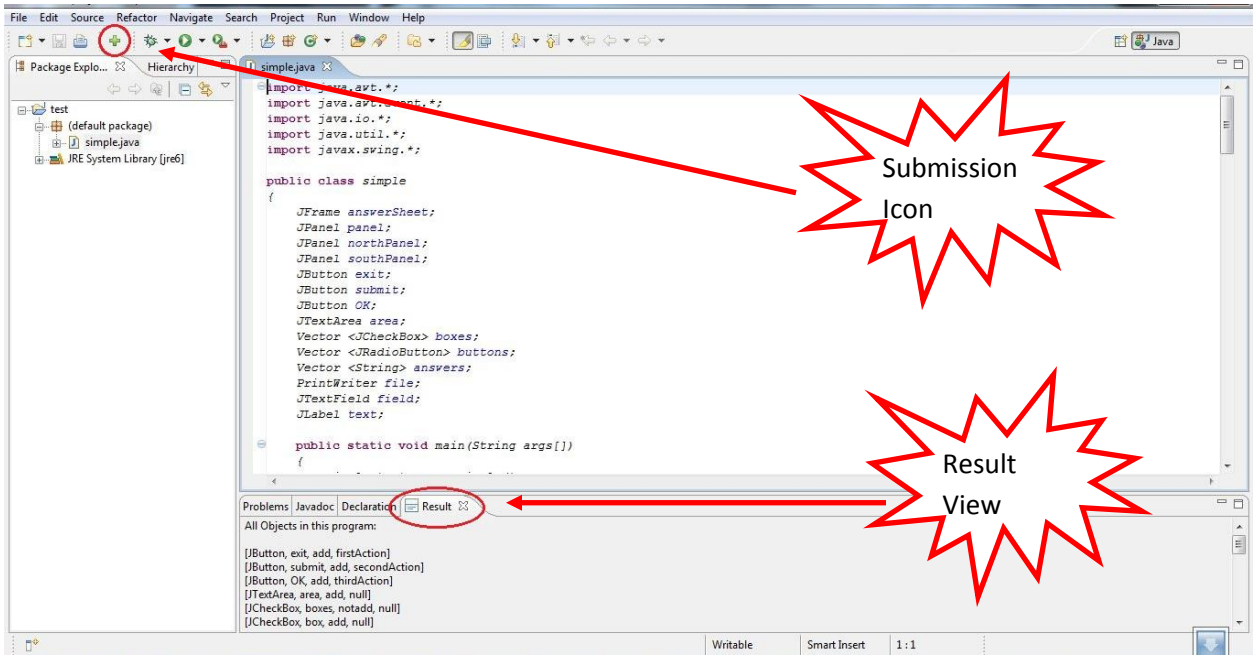

Figure 4: JavaMarker Plugin

4.2 GUI Assessment Tools

Sun and Jones [14] In their first attempt introduced an approach for marking graphical user interfaces in which a specific program specification provided to students. For example if a program is to enter a data in a *textfiled* and process it by clicking a *button*, students must follow the same design. Also it is not possible to have more than one window or to define objects with different names from what is said in the program specification.

GUI_Grader [6] is another automated tool that lets students to define more than one window. It also allows students to choose the objects they want to use for their program. With this option, they allow more flexibility in designing the interface than what is implemented in [14].

Another well developed graphical user interface marker is what is reported in [7]. In this system an introspective approach was applied to look into students' program. It gives freedom to students to choose the kinds of the objects that they want while setting restriction is also possible. This system was developed using *Dynamic Class Loading* [11]. A XML-based test case in which various actions that should be defined on each component is provided by teaching staff.

5. CONCLUSION & FUTURE WORK

In this paper the development of a making system for graphical interfaces as a plugin for Eclipse was introduced. Since this is an early stage of the work, some works can be carried out to improve the system. These improvements are listed below.

Dynamic Class Loading [11] is a facility that is provided by object oriented languages such as Java. This can be used for automated marking of graphical user interfaces with more flexibility than what is discussed in this paper. With this facility we can allow students to be more creative and not to be dependent strictly to what is set in program specification for them. Therefore the next generation of this plugin will be the improved version of the code using *Class Loader* facility. To use this facility one should take into consideration that the security of system might be compromised. Java *Class Loader* loads every class, which means it gives the same privilege to every one of the classes. This opens an opportunity for a malicious code to attack the system. In case the program was upgraded using this facility, this issue must be taken seriously.

As previously explained this plugin and its ascendant will be integrated. To do so the current plugin should be able to run at client side while all the repositories reside on server side. At the moment for the complex nature of this program we decided to keep everything locally on client machines. Also test case files will join the database that had been defined previously for text-based version of JavaMarker. This database resides in server, which its replacement will take place at integration time.

6. REFERENCES

[1] Ahmadzadeh, M., Namvar, S., Solatani, M., JavaMarker: 2011, A Marking System for Java Programs, International Journal of Computer Applications, Volume 20– No.2, April 2011. P 15-20.

[2] Ahmadzadeh, M., Soltani, M. 2010, JavaMarker: An Eclipse Plug-in to mark Students' Java Exercises. ITiCSE '10: Proceedings of the 2010 ACM SIGCSE Annual Conference on Innovation and Technology in Computer Science Education. Ankara, Turkey. June 26-30, p 324.

[3] Benford, S., Burke, E., Foxley, E. and Higgins, C. 1995. The Ceilidh system for the automatic grading of students on programming courses. In *Proceedings of the 33rd annual southeast regional conference* (ACM-SE 33). ACM, New York, NY, USA, 176-182.

[4] Daly, C. AND Waldron, J. 2004. Assessing the assessment of programming ability. In Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education. 210-213.

[5] Daly, C. 1999 RoboProf and an Introductory Computer Programming Course. In Proceedings of the 4th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education. 155-158.

[6] Feng, M.Y., McAllister A., 2006. A Tool for Automated GUI Program Grading, 36th ASEE/IEEE Frontiersin Education Conference, October 28-31, San Diego, USA.

[7] Gray, G. R., Higgins, C. A. 2006. An Introspective Approach to Marking Graphical User Interfaces. ITiCSE '06: Proceedings of the 2006 ACM SIGCSE Annual Conference on Innovation and Technology in Computer Science Education. Bologna, Italy, June 26-30.

[8] Jansen, B.J., 1998, The Graphical User Interface: An Introduction, SIGCHI Bulletin pp22-26.

[9] Joy, M., Griffiths, N., and Boyatt, R., 2005.The BOSS Online Submission and Assessment System., ACM Journal on Education Resources in Computing, vol.5, No.3, September 2005, Article 2.

[10] Higgins, C., Hegazy, T., Symeonidis, P., AND Tsintsifas, A. 2003. The CourseMaster CBA system:Improvements over Ceilidh. J. Edu. Inf.Technol. 8, 3, 287-304.

[11] Liang, S., Bracha, G. 1998, Dynamic Class Loading in the Java TM Virtual Machine, Proceeding of 13th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications, Vancouver, BC, October 1998.

[12] Myers, B.A. 1993, Why are Human-Computer Interfaces Difficult to Implement? Technical Report CMU-CS-93-183, Carnegie Mellon University, Pittsburgh, July 1993.

[13] Reek, K. A. 1989. The TRY system – or – how to avoid testing student programs. SIGCSE Bull. 21, 1, 112-116.

[14] Sun, H., Jones, E.L. 2004, Specification-driven Automated Testing of GUI-based Java Programs, Proceedings 42nd ACM Southeast regional Conference, Huntsville, Alabama, USA, April 2-3, pp. 140-145

[15] Sun Microsystem. April 20, 1999. Code Conventions for the Java Programming Language. http://www.oracle.com/technetwork/java/codeconventions-150003.pdf. Last Accessed on 6/1/2011