A Tool for Automated GUI Program Grading

Man Yu Feng and Andrew McAllister
University of New Brunswick, Faculty of Computer Science,
Fredericton, Canada, matthewfeng@gmail.com, andrewm@unb.ca

Abstract - **This paper introduces an automated grader for Java programs called GUI_Grader that allows students a degree of flexibility in graphical user interface (GUI) design. GUI_Grader allows students to build multi-window Java applications, choose among alternative GUI components, and decide how to order, position and label components. This enables students to practice some aspects of designing their own GUI applications while still providing automated grading `based on a single test plan. The data-driven approach helps to maintain consistency between test plans and program specifications. Testing GUI_Grader on Java assignments from both a first-year and an upper-year course confirms the feasibility of the approach.**

Index Terms – Automated grading, GUI design, Programming specifications, Software testing.

INTRODUCTION

The manual labor associated with grading student assignments is a common problem for computer programming courses. A number of automated grading systems have been developed to address this issue, the majority of which are restricted to marking programs with textual input and output [1, 5, 6, 8, 10, 11] or to evaluating Excel spreadsheets and Access database designs [3]. Automated grading of student programs with a graphical user interface (GUI) involves the additional issue of simulating user interaction with GUI components such as text fields and buttons.

Sun and Jones [12] describe a first attempt to define an approach for automated grading of Java GUI programs. In this approach, a course instructor provides very specific instructions to students about how their GUI must be designed. The instructor also provides an XML-based test specification, which defines test cases that include various actions to be performed on specific GUI components. For example, a test case might simulate entering a value in a text field, clicking a button to get the student's program to perform a calculation, then comparing the value that appears in a result text field with the expected result. This approach places a number of restrictions on the GUI programs written by students. Each student's GUI must:

- consist of only a single window (i.e. one JFrame instance); and
- include exactly the set of Java object types specified in the assignment specification, in the same order and with the same names as in the specification.

This paper introduces an automated grader for Java GUI programs called GUI_Grader that relaxes these restrictions. GUI_Grader allows students to:

- build applications with multiple windows, including both JFrame and modal dialog windows;
- choose which types of Java objects they wish to use to satisfy the needs specified by an assignment;
- position these objects within their GUI layouts in an order of their own choosing; and
- choose appropriate labels for various GUI objects.

A variety of tools are available for testing GUI applications, including commercial products (e.g. Mercury QuickTest [7], Rational Robot [4]) as well as open source (e.g. Costello based on Abbot [2]). Using such tools, testers develop scripts for a given implementation of a system. Since scripts contain references to specific GUI components, they typically cannot be used for multiple applications whose components may differ. A key requirement for grading assignments is to test multiple applications whose user interfaces differ in significant ways, and yet use a single test specification.

The simple calculator programs shown in Figures 1 and 2 illustrate typical ways in which student solutions can differ. Both programs allow the user to enter two numeric values, select one of four arithmetic operations and click a button to see a result dialog. Tom used a group of radio buttons for selecting the operation, while Jack used a drop-down combo box. The components are also arranged and labeled differently within the two windows. GUI_Grader is able to grade both programs based on a single program specification and a single set of test data.

There are significant pedagogical issues behind the need for such a grading tool. First, students tend to progress rather rapidly to the point where they are creating multi-window applications, even in first-year programming courses. Second, programming assignments often involve other objectives besides gaining experience with writing and testing source code. Students should also be given opportunities to learn to select appropriate user interface components and to design effective GUI layouts. GUI_Grader allows students to make some of these decisions on their own, as opposed to forcing all student solutions to conform to a common, fully-specified design.

This paper is organized as follows. First we provide an overview of the GUI_Grader architecture. Following this, the next two sections show how assignment data are organized,

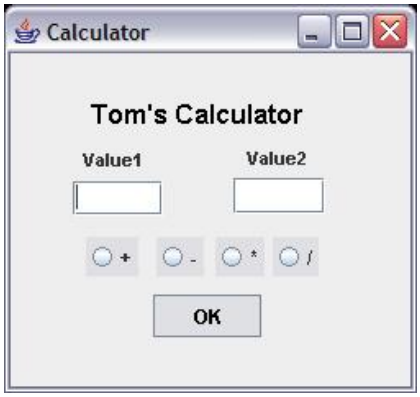

FIGURE 1
FIRST CALCULATOR EXAMPLE.

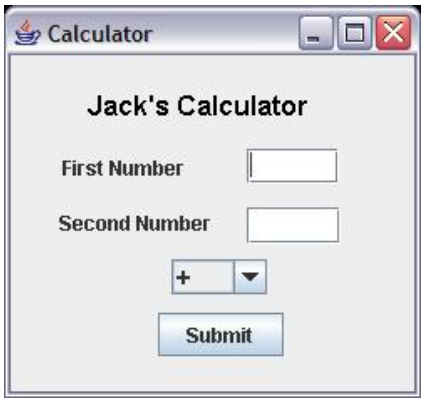

FIGURE 2
SECOND CALCULATOR EXAMPLE.

focusing on assignment specifications and test data, respectively. This data organization represents the key to how GUI_Grader supports flexibility in GUI design. The final section discusses conclusions and future work.

GUI_GRADER OVERVIEW

GUI_Grader employs a data-driven architecture, as shown in Figure 3. The assignment definition database includes two types of data: assignment specifications and test cases. The assignment maintenance tool provides a set of GUI forms that allow instructors to create and edit an assignment specification and then create a set of test cases that will be used to grade the programs completed for this assignment.

Not every student is skilled at interpreting the raw contents of database tables. The assignment formatting tool converts raw assignment specification data into a readable form for students. An assignment specification defines:

- the windows that the assigned program should include;
- the types of GUI components that each window should include (with some flexibility allowed, as illustrated by Figures 1 and 2); and
- the required functionality.

For example, the assignment specification for the calculator application indicates that the program must include a single JFrame window with two JTextField objects, a GUI object that allows the user to choose one of four operations, and a JButton. The application must also include a dialog window to show the results of calculations, as well as a few other dialogs for error messages (e.g. if one of the text fields contains a non-numeric value when the button is clicked). The functional specification defines what should happen when the button is clicked, including the precise conditions under which each of the error message dialogs should appear.

Based on the assignment specification, each student writes and debugs a solution, then submits it to the assignment submission repository. Our students use a WebCT application for electronic submission of assignment solutions. WebCT predates and is entirely independent from GUI_Grader. By submitting the appropriate requests to the WebCT interface, GUI_Grader retrieves each assignment submission in turn, then uses the assignment specification data combined with the test cases to grade each one. Grade information is provided to both the students and the instructors in the form of grade reports, as described below.

GUI_Grader uses the NetBeans Jemmy library [9] to interact with Java programs written by students. The Jemmy

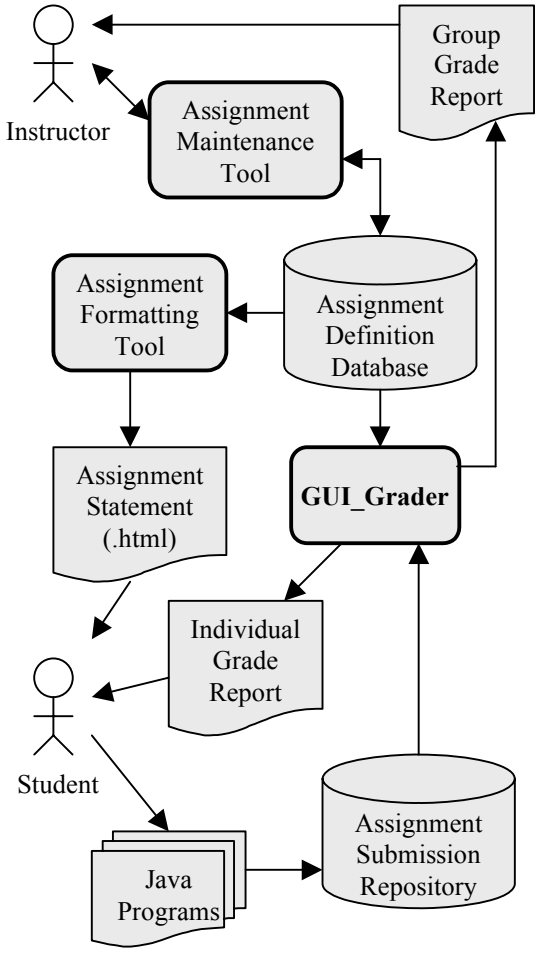

FIGURE 3
GUI_GRADER ARCHITECTURE.

methods simulate user interaction with GUI objects, such as clicking buttons or menu items, entering text into fields and verifying the behavior of the student software.

The existing automated graders mentioned above provide functionality similar to some parts of Figure 3. These existing solutions allow students to submit solutions electronically, execute each program in turn by applying a series of test cases, and provide feedback about the resulting grades. The primary innovations of GUI_Grader are found within the assignment definition database, along with the way GUI_Grader uses the data to feed a single set of test data to programs with differing components. This paper highlights these innovations by focusing on the database organization.

ASSIGNMENT SPECIFICATIONS

The assignment definition database defines the GUI components that students must include in their calculator programs. Table I shows the contents of the WINDOW database table to define the four windows required for this program. "JFrame" refers to a Java class included in the Swing library, while the term "Dialog" means that students are to use Java's JOptionPane class to create a modal dialog. Students are free to use whatever JOptionPane method they wish to achieve the desired results.

GUI_Grader supports multiple assignments within multiple sections of multiple courses. To simplify discussion in the context of this paper, however, id values for courses, sections and assignments are omitted from the database tables.

Table II provides additional information about the types of components that must appear in each window. The *Component Base Id* field defines a name for each component. GUI_Grader uses these names to access the components, so for each JFrame window component defined in WINDOW_COMPONENT, students must include a statement in their program that invokes the *setName* method to associate the specified name with that component. For example, the button in Figure 1 might be coded as follows, consistent with row 4 of Table II:

```
JButton submitButton = new JButton("OK");
submitButton.setName("Submit");
```

JOptionPane dialogs are created by method calls, so students can't access the window components directly to use setName. Instead, GUI_Grader uses the default component names of the form JOptionPane. <ComponentClassName>, as shown in the last six rows of Table II. This works fine for most JOptionPane components (e.g. labels, icons, text fields) because there is normally only one component of a given type per dialog. This is not true, however, for buttons. It is common, for instance, to have a dialog with buttons labeled YES, NO and CANCEL. For this reason, GUI_Grader uses label text to identify JButton objects on JOptionPane dialog windows. The last three rows of Table III specify the button labels for the Calculator dialogs.

TABLE I
WINDOW

Window Id	Window Type
Calculator	JFrame
Result	Dialog
InvalidInput	Dialog
InvalidDivisor	Dialog

TABLE II
WINDOW_COMPONENT

Window Id	**Component Base Id**	**Component Type**
Calculator	Value1	Text Input
Calculator	Value2	Text Input
Calculator	Operation	Single Item Selection
Calculator	Submit	JButton
Result	JOptionPane.JLabel	JLabel
Result	JOptionPane.JButton	JButton
InvalidInput	JOptionPane.JLabel	JLabel
InvalidInput	JOptionPane.JButton	JButton
InvalidDivisor	JOptionPane.JLabel	JLabel
InvalidDivisor	JOptionPane.JButton	JButton

The COMPONENT_OPTION database table shown in Table III is also used to define the selection options for GUI components such as JComboBox objects.

TABLE III
COMPONENT_OPTION

Window Id	**Component Base Id**	**Option Id**
Calculator	Operation	+
Calculator	Operation	-
Calculator	Operation	*
Calculator	Operation	/
Result	JOptionPane.JButton	OK
InvalidInput	JOptionPane.JButton	OK
InvalidDivisor	JOptionPane.JButton	OK

When designing the GUI for an assignment, instructors can insist that specific Java classes are to be used, or they may give students some flexibility in choosing which classes to use. The Component Type column of Table II indicates that students are to use specific Java classes (e.g. JLabel, JButton) for several components. Terms like *Text Input* and *Single Item Selection* give students a degree of choice, as indicated by Table IV. For example, a student can use either a JTextField or a JTextArea for a Text Input component.

TABLE IV
FLEXIBLE GUI COMPONENT TYPES

Flexible Type Name	**Java Swing Classes**
Single Item Selection	JRadioButton group, JComboBox
Multiple Item Selection	JCheckBox, JList
Text Input	JTextField, JTextArea,
Text Display	JLabel, JTextField, JTextArea
Event Selection	JButton, JMenuItem

Tables I, II and III define the minimum mandatory components for each student program, which GUI_Grader expects to access when executing test cases. Students are also expected to include other GUI components, such as the labels for the text fields in Figures 1 and 2. GUI_Grader does not automatically assess these "extra" components, but instructors

can choose to manually assess such program attributes if they wish, along with others such as code formatting, comments and visual appeal of the user interface. In this case, the grade supplied by GUI_Grader forms a part of the overall assignment grade and relieves the marker from the manual effort associated with verifying correct functionality.

The final part of the assignment specification is the functional specification. This is a textual description of how the software should perform, which includes references to the GUI components defined in Tables I, II and III, as shown in Figure 4.

> Write a Java program for a simple calculator. In the Calculator window, the user enters a numeric value in each of the Value1 and Value2 fields, selects one of four operations (+, -, * or /) using the Operation component, then clicks the Submit button. The Result window appears, displaying in the JOptionPane.JLabel component a message that includes the numeric result of the operation.
>
> The InvalidInput window appears if the user clicks the Submit button when either Value1 or Value2 are empty or contain a non-numeric value, or when no operation is selected. Display an appropriate message in the JLabel component.
>
> The InvalidDivisor window appears if the user clicks the Submit button when the / operation is selected, Value1 contains a valid numeric value and Value2 contains 0 (i.e. the numeric value zero). Display an appropriate message in the JLabel component that includes the substring "zero".

FIGURE 4
EXAMPLE FUNCTIONAL SPECIFICATION.

With the current implementation of the tool, instructors load the assignment specification database tables manually with the data described above. We are currently building an Assignment Maintenance Tool that provides a GUI for entering such data. For example, when specifying the types of windows that students must include in their assignment solutions, the Assignment Maintenance Tool presents a drop-down list to the instructor, allowing selection of either JFrame or Dialogue as the window type.

The Assignment Formatting Tool mentioned in Figure 3 provides the functional specification along with all the information contained in Tables I to III to students in a somewhat more readable format (as compared with simply providing the raw data in the tables). Figure 5 provides a partial example of this format based on the specification data discussed above. Formatted assignments can be viewed online or printed by students.

TEST CASES

A test plan consists of a set of numbered test cases (Table V), each of which includes one or more actions (Table VI). The test case *Description* field indicates the purpose of each test case. These descriptions are used in the grading report provided to students so they will understand which functions their program was able to perform successfully and which functions failed. The instructor also assigns each test case a number of "points". This determines what the entire assignment will be marked out of (the points total for all test cases), and the relative weight for each test case.

> This application includes 1 JFrame window and 3 dialogs.
>
> Calculator is a JFrame window that includes the following mandatory components:
>
> | | |
> |---|---|
> | Value1: | Text Input (JTextField or JTextArea) |
> | Value2: | Text Input (JTextField or JTextArea) |
> | Operation: | Single Selection (JRadioButton or JComboBox) with Options: +, -, *, / |
> | Submit: | JButton |
>
> *(etc.)*

FIGURE 5
FORMATTED PARTIAL ASSIGNMENT.

TABLE V
TEST_CASE

Test Case #	Description	Points
1	Addition operation	2
2	Division by zero	2

To execute a test case, GUI_Grader performs each action in turn. For each action, GUI_Grader first uses Jemmy library methods to verify that a GUI component with the specified name (*Component Base Id* in Table VI) and an appropriate type (as per Table II) exists for the specified window. If not, the test case fails and GUI_Grader moves on to the next test case. If the component is found, an action of the specified type is performed by invoking appropriate Jemmy methods. For example, test case 1 in Table VI gets the calculator to add 1 plus 2, verifies that the label on the Result dialog includes the digit 3, and verifies that the Result dialog includes the appropriate button.

The six action types supported by GUI_Grader are listed in Table VII. These simulate user interaction with the GUI components.

An *Entry* action simulates the user pressing any keyboard key, using a list of keywords specific to GUI_Grader. Examples include "F1", "Tab" and "Page Up". *Contains* and *Equals* actions represent verification of expected test case results.

The grade report produced for each student program summarizes the results of each test case. Table VIII shows an example report for a calculator program that was able to add 1 and 2 correctly but failed to produce an appropriate message dialog for division by zero.

TABLE VI
TEST_ACTION

Test Case #	**Action #**	Window Id	Component Base Id	Action Type	Value
1	1	Calculator	Value1	Input	1
1	2	Calculator	Value2	Input	2
1	3	Calculator	Operation	Select	+
1	4	Calculator	Submit	Click	null
1	5	Result	JOptionPane.JLabel	Includes	3
1	6	Result	JOptionPane.JButton	Click	OK
2	1	Calculator	Value1	Input	4
2	2	Calculator	Value2	Input	0
2	3	Calculator	Operation	Select	/
2	4	Calculator	Submit	Click	null
2	5	Result	JOptionPane.JLabel	Includes	zero
2	6	Result	JOptionPane.JButton	Click	OK

TABLE VII
ACTION TYPES

Action Type	Type of Value	Action Description
Input	String	Input to a Text Input component
Entry	Keyword	A keyboard entry: e.g. Pressing the "Esc" key
Select	Option Id	Selecting an option of a Single or Multiple Item Selection
Click	null or caption text	A mouse click on a button or menu item
Contains	String	Verifying if the String is a substring of the text in the component
Equals	String	Verifying if the String equals the text in the component

TABLE VIII
EXAMPLE GRADE REPORT

Test Case #	Description	Failed in Step #	Points	Out of
1	Addition operation	n/a	2	2
2	Division by zero	5	0	2
		TOTAL:	2	4

CONCLUSIONS AND FUTURE WORK

To confirm the soundness of this approach, we enlisted the help of four student volunteers to provide sample solutions for two representative assignments previously used in our Java programming courses. One was from a first-year introductory programming course and the other was from an upper-year software engineering course. This involved redefining the original assignment specifications to conform to GUI_Grader's naming and flexible type conventions, which turned out to be a straightforward exercise. As compared with our traditional way of publishing informal specifications for the students, the extra work required to define an assignment for GUI_Grader is minimal—less than an hour per assignment for the examples we tested.

Similarly, the student volunteers who produced sample programs for these assignments reported no difficulties in learning and applying the concepts of component naming and of flexible GUI component types. These students found that inserting setName calls in their programs involved an insignificant amount of effort.

For each assignment, the solutions provided differed considerably in appearance and GUI design. Nonetheless, GUI_Grader was able to produce accurate grade reports (compared with manual evaluation) for all programs based on a single set of test data.

By allowing multi-window programs and relaxing the restrictions on students' GUI designs, GUI_Grader offers significant improvements over the current state of the art in automated grading for GUI programs. In addition, test data in the GUI_Grader database are linked directly to the GUI components defined in the corresponding assignment specification tables. Referential integrity ensures that the test data values are consistent with the specification. For example, the database disallows a test case that refers to nonexistent GUI components. This type of consistency is maintained regardless of whether the software specification and test plan are developed by the same person or by different people. Based on informal discussion, the student volunteers had no difficulty understanding this concept of consistency within the specifications.

There are several opportunities for future work, the most significant of which is to relax the restrictions on GUI design even further. The more students are able to make their own decisions, the more design experience they have the opportunity to gain. We are currently investigating how to allow students to decide what windows their program will include and on which window each required GUI component will reside.

Our work so far has focused on handling flexible GUI designs; other aspects of the environment are still relatively simplistic. For instance, evaluation of expected results can be enhanced to handle more complex analysis of textual output fields, perhaps based on regular expressions or numeric ranges. A more sophisticated scheme for assigning part marks is also possible. For example, the TEST_ACTION table might specify that a program that successfully completes a given

action partway through a test case should receive another mark. This is less of an "all or nothing" approach.

ACKNOWLEDGMENT

The research reported in this paper was supported by a Discovery Grant from the Canadian National Sciences and Engineering Research Council.

REFERENCES

[1] Cheang, B., Kurnia, A., Lim, A. and Oon, W.-C. "On automated grading of programming assignments in an academic institution", *Computers & Education*, Vol. 41, 2003, pp. 121-131.

[2] Costello. Getting started with the Abbott Java GUI test framework. http://abbot.sourceforge.net/doc/overview.shtml, accessed May 2006.

[3] Hill, T.G. "Excel grader and Access grader", *SIGCSE Bulletin*, Vol. 36, No. 2, June 2004, pp. 101-105.

[4] IBM. Rational Robot homepage. http://www-306.ibm.com/software/awdtools/tester/robot/, accessed May 2006.

[5] Jackson, D. and Usher, M. "Grading student programs using ASSYST", *Proceedings 28th SIGCSE Technical Symposium on Computer Science Education*, San Jose, California, USA, February 27-March 1, 1997, pp. 335-339.

[6] Jones, E.L. "Grading student programs - a software testing approach", *Journal of Computing in Small Colleges*, Vol. 16, No. 2, January 2001, pp. 185-192.

[7] Mercury. QuickTest homepage. http://www.mercury.com/us/products/quality-center/functional-testing/quicktest-professional/, accessed May 2006.

[8] Morris, D.S. "Automatic grading of student's programming assignments: An interactive process and suite of programs", *Proceedings 33rd ASEE/IEEE Frontiers in Education Conference*, Boulder, Colorado, USA, November 5-8, 2003, pp. 112-116.

[9] NetBeans. Jemmy homepage. http://jemmy.netbeans.org, accessed May 2006.

[10] Reek, K.A. "The TRY system - or- how to avoid testing student programs", *ACM SIGCSE Bulletin*, 21, 1, 1989, pp. 112-116.

[11] Reek, K.A. "A software infrastructure to support introductory computer science courses", *Proceedings 27th SIGCSE Technical Symposium on Computer Science Education*, Philadelphia, USA, February 15-17, 1996, pp. 125-129.

[12] Sun, H. and Jones, E.L. "Specification-driven automated testing of GUI-based Java programs", *Proceedings 42nd ACM Southeast Regional Conference*, Huntsville, Alabama, USA, April 2-3, 2004, pp. 140-145.