Automated grading of student-designed GUI programs

Andrew McAllister & Man Yu Feng

University of New Brunswick
Fredericton, Canada

ABSTRACT: In this article, the authors introduce an automated grader for Java™ programs, called GUI_Grader, which allows students a degree of flexibility in graphical user interface (GUI) design. A previously published solution limits student programs to a single window and forces instructors to make virtually all GUI design decisions. GUI_Grader allows students to build multi-window Java applications, choose among alternative GUI components, and decide how to order, position and label the components. Compared with other automated graders, this approach can be used further into the curriculum and supports important GUI design learning objectives. The data-driven approach also helps to maintain consistency between program specifications and test plans. Testing GUI_Grader on programming assignments from second-term courses confirms the usability of the approach.

INTRODUCTION

The manual labour associated with grading student assignments is a common problem for computer programming courses. A number of automated grading systems have been developed in order to address this issue, the majority of which are restricted to marking programs with textual input and output [1-6]. Another evaluates *Excel* spreadsheets and *Access* database designs [7]. The automated evaluation of student programs with a graphical user interface (GUI) is more problematic because of the need to simulate user interaction with GUI components, such as text fields and buttons.

Sun and Jones describe a first attempt to define an approach for the automated grading of Java GUI programs [8]. 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.

Sun and Jones' approach places a number of restrictions on the GUI programs written by students. Each student's GUI must:

- Consist of only a single window (ie one JFrame instance);
- 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.

In this article, the authors introduce an automated grader for Java GUI programs, called GUI_Grader, which relaxes these restrictions and thus supports a wider variety of learning opportunities for students. GUI_Grader allows students carry out the following:

- Build applications with multiple windows, including both JFrame and modal dialogue 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;
- Choose the appropriate labels for various GUI objects.

The simple calculator programs shown in Figures 1 and 2 illustrate some of these types of flexibility. Both programs allow the user to enter two numeric values, select one of four arithmetic operations and click a button to see a result dialogue. In Figure 1, Tom used a group of radio buttons for selecting the operation, while in Figure 2, Jack used a drop-down combo box. The components are also arranged and labelled 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 beyond simple single-window applications, even in first-year computer science courses. Text-only user interfaces are common in introductory courses but tend to be used less frequently in upper years. The authors' approach enables automated grading to be used further into the curriculum.

Second, programming assignments often involve other objectives besides writing and testing source code. Students must also learn to select appropriate user interface components and to design effective GUI layouts. GUI_Grader provides the flexibility to support these learning goals.

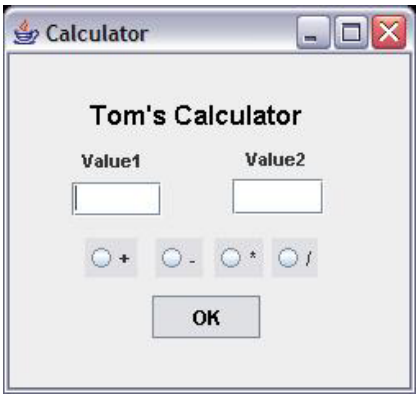

Figure 1: First calculator example.

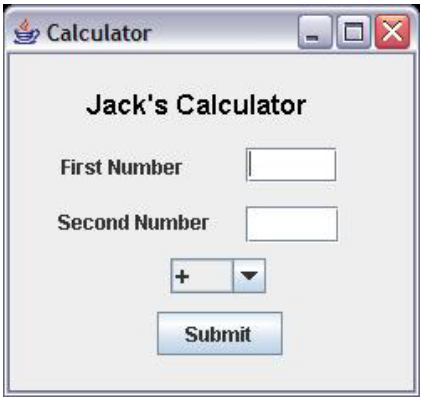

Figure 2: Second calculator example.

This article is organised as follows: an overview of the GUI_Grader architecture is provided, followed by a presentation of how the assignment data are organised, focusing on assignment specifications and test data, respectively. This data organisation represents the key to how GUI_Grader supports flexibility in GUI design. The last section presents 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.

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);
- 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 dialogue window to show the results of calculations, as well as a few other dialogues for error messages (eg 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 dialogues should appear.

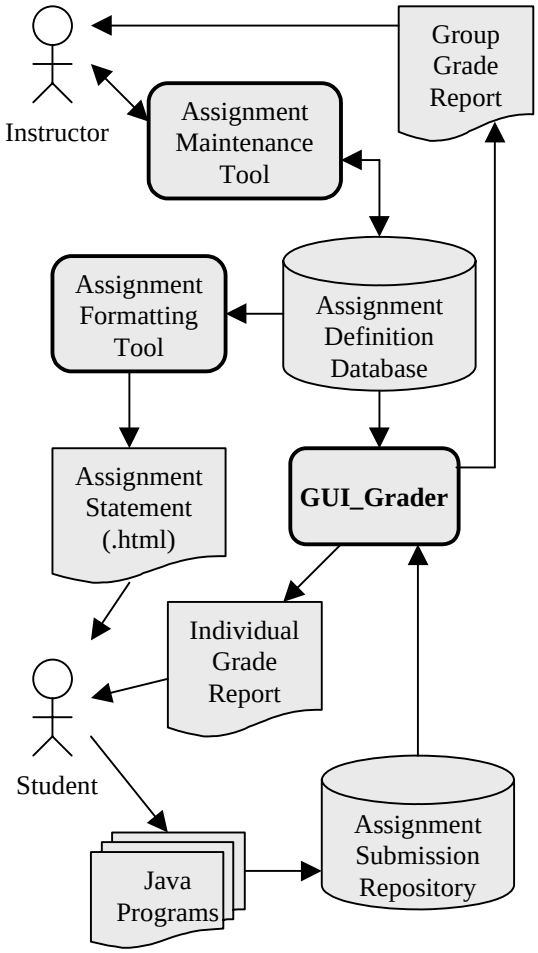

Figure 3: GUI_Grader architecture.

Based on the assignment specification, each student writes and debugs a solution, then submits it to the assignment submission repository. GUI_Grader uses the assignment specification data combined with the test cases to grade each submission in turn, and provides grade information to both the students and instructors.

Similar to Sun and Jones, GUI_Grader uses the NetBeans Jemmy library to interact with Java programs written by students [9]. The Jemmy methods simulate user interaction with GUI objects, such as clicking buttons or menu items, entering text into fields and verifying the behaviour of a student's software.

The existing automated graders mentioned in the first section 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 can be 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. The authors highlight these innovations by focusing on the database organisation.

ASSIGNMENT SPECIFICATIONS

The assignment definition database defines the GUI components that students must include in their calculator programs. Table 1 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 *Dialogue* means that students are to use Java's JOptionPane class to create a modal dialogue. Students are free to use whatever JOptionPane method they wish to achieve the desired results.

Table 1: WINDOW.

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

Table 2 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 2:

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

Table 2: 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

JOptionPane dialogues are created by method calls, so students cannot 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 2. This works fine for most JoptionPane components (eg labels, icons, text fields), because there is normally only one component of a given type per dialogue. However, this is not true for buttons.

It is common, for instance, to have a dialogue with buttons labelled YES, NO and CANCEL. For this reason, GUI_Grader uses label text to identify JButton objects on JOptionPane dialogue windows. The last three rows of Table 3 specify the button labels for the Calculator dialogues.

The COMPONENT_OPTION database table shown in Table 3 is also used to define the selection options for GUI components such as JComboBox objects.

Table 3: 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 in Table 2 indicates that students are to use specific Java classes (eg JLabel, JButton) for several components. Terms like *Text Input* and *Single Item Selection* give students a degree of choice, as indicated by Table 4. For example, a student can use either a JTextField or a JTextArea for a *Text Input* component.

Table 4: 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 1, 2 and 3 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.

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 1, 2 and 3, as shown in Figure 4.

TEST CASES

A test plan consists of a set of numbered test cases (see Table 5), each of which includes one or more actions (see Table 6). 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.

Table 6: 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

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 (ie the numeric value zero). Display an appropriate message in the JLabel component that includes the substring *zero*.

Figure 4: Example of functional specification.

Table 5: 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 6) and an appropriate type (as per Table 2) 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 6 gets the calculator to add 1 plus 2, verifies that the label on the Result dialogue includes the digit 3, and verifies that the Result dialogue includes the appropriate button.

The six action types supported by GUI_Grader are listed in Table 7. 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.

Table 7: Action types.

Action Type	Type of Value	Action Description
Input	String	Input to a Text Input component
Entry	Keyword	A keyboard entry, eg 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

A grade report produced for a student program is simply a summary of the results of each test case. Table 8 shows an example report for a calculator program that was able to add 1 and 2 correctly, but failed to produce an appropriate message dialogue for division by zero.

Table 8: 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

The soundness of this approach has been confirmed by testing GUI_Grader with three multi-window GUI assignments previously used in the second-term Java programming course. This involved redefining the original assignment specifications somewhat to conform to GUI_Grader's naming and flexible type conventions, which turned out to be a straightforward exercise. As compared with the traditional way of publishing informal specifications for students, the extra work required to define an assignment for GUI_Grader is minimal – less than an hour per assignment for the examples that were tested.

Similarly, the first-year student volunteers who produced sample programs for these assignments found that inserting setName calls in their programs involved an insignificant

amount of effort. These students reported no difficulties in understanding and applying the concept of flexible GUI component types. GUI_Grader was able to produce accurate grade reports (compared with a manual evaluation) for all programs.

By allowing multi-window programs and relaxing the restrictions on students' GUI designs, GUI_Grader offers significant improvements over the approach employed by Sun and Jones [8]. In addition, using a database to store assignment specifications and test data provides another advantage. In Sun and Jones, the software specification for a given assignment is stored separately from the textual (XML) specification of the test cases [8]. There is no automated support to ensure consistency between the specification and the test plan. Conversely, test data in the authors' database is linked directly to the GUI components defined in the corresponding assignment specification. Referential integrity ensures that the test data is 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.

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 gain and the further into the curriculum GUI_Grader can be used. The authors 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.

The authors' work has so far focused on handling flexible GUI designs; other aspects of the environment are still relatively simplistic. For instance, an evaluation of expected results can be enhanced to handle a 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.

Finally, it is anticipated that future practical experience using GUI_Grader with various computer science courses will enable educators to refine the approach to support courses at several levels of the curriculum.

REFERENCES

1. Cheang, B., Kurnia, A., Lim, A. and Oon, W-C., On automated grading of programming assignments in an academic institution. *Computers & Educ.*, 41, 121-131 (2003).
2. Jackson, D. and Usher, M., Grading student programs using ASSYST. *Proc. 28th SIGCSE Technical Symp. on Computer Science Educ.*, San José, USA, 335-339 (1997).
3. Jones, E.L., Grading student programs - a software testing approach. *J. of Computing in Small Colleges*, 16, **2**, 185-192 (2001).
4. Morris, D.S., Automatic grading of student's programming assignments: an interactive process and suite of programs. *Proc. 33rd ASEE/IEEE Frontiers in Educ. Conf.*, Boulder, USA, 112-116 (2003).
5. Reek, K.A., The TRY system – or how to avoid testing student programs. *ACM SIGCSE Bulletin*, 21, **1**, 112-116 (1989).
6. Reek, K.A., A software infrastructure to support introductory computer science courses. *Proc. 27th SIGCSE Technical Symp. on Computer Science Educ.*, Philadelphia, USA, 125-129 (1996).
7. Hill, T.G., *Excel* grader and *Access* grader. *SIGCSE Bulletin*, 36, **2**, 101-105 (2004).
8. Sun, H. and Jones, E.L., Specification-driven automated testing of GUI-based Java programs. *Proc. 42nd ACM Southeast Regional Conf.*, Huntsville, USA, 140-145 (2004).
9. NetBeans, Jemmy homepage, http://jemmy.netbeans.org