Automated Grading Systems for Programming Assignments: A Literature Review

Hussam Aldriye[1], Asma Alkhalaf[2], Muath Alkhalaf[3]
Computer Science at the Collage of Computer, King Saud University, Riyadh, Saudi Arabia[1, 3]
Computer Science at Alrass Collage of Science and Arts, Qassim University, Qassim, Saudi Arabia[2]

***Abstract*—Automated grading for programming assignments is becoming more and more important nowadays especially with the emergence of the Massive Open Online Courses. Many techniques and systems are being used nowadays for automated grading in educational institutions. This article provides a literature review of the many automated grading systems and techniques that are being used currently. It focuses on highlighting the differences between these systems and techniques and addressing issues, advantages and disadvantages. The review shows that these systems have limitations due to difficulty in usage by students as noticed by some course instructors. Some of these problems stem from UI/UX difficulties while other problems were due to beginner syntax errors and language barriers. Finally, it shows the need to fill the gap by building new systems that are friendlier towards beginner programmers, has better localization and easier user experience.**

Keywords—Automated grading

I. Introduction

Innovation in education has come a long way in improving the speed and efficiency of grading. For example, the inventiveness behind the idea of the Scantron has significantly promoted the grading of tests for instructors and has consequently provided a better means for professors to check on students' knowledge of particular concepts. Homework and exams are the best way to determine student comprehension. However, grading programming assignments is time consuming and prone to errors especially with large number of students in the Massive Open Online Courses (MOOS) and complex programming assignments. This raises the need for a more consistent and efficient grading technique. This challenge has lead to the development of automated grading tools. This paper provides comparison and evaluation of different tools used for automated grading for programming assignments focusing on the effectiveness of these tools in learning process.

This article is organized as follows. Section 2 describes the software defects while Section 3 presents the literature review of the current automated grading techniques and the tools applying those techniques. Section 4 provides summary comparison of the reviewed tools. Finally, Section 5 concludes the article.

II. Software Defects

Before we review the literature on automated assignment grading techniques and systems, we would like to introduce the types of errors that are usually targeted by these techniques. In general, a software defect, failure or error is defined as producing wrong result or performing an action in an unintended way. However, Software defects can be classified as following syntax errors, logical errors, and runtime errors.

1) Syntax error: To This error is raised due to incorrect grammar/syntax in the programming language such as incorrect program structure, mistyped words (typos), missing semicolons. Moreover, this kind of errors can be detected by the programming language compiler while compiling the software code. This error is the easiest error to catch and fix since most of the compilers that used this day such as GCC or JRE provides a full description of the error (line number and message show what is missing).

2) Logical errors: In this error, the software compiles and runs fine, but the output of the software is wrong due to many reasons such as misunderstanding of the requirement or specification, logical-mathematical errors (divide by zero, adding when you should be subtracting) and opening and using data from the wrong source. These errors, unfortunately, cannot be detected by the compiler and this issue brings up a question, can we detect this kind of errors before launching the software? The short answer is yes, by using testing methods and other techniques, this will be described in detail in Section 2.2.

3) Runtime errors: This is an advanced error, and it is rare to introductory course students to fall in. Runtime error will only happen when the software is running. In fact, this is one of the most complicated issues to track down and lead to software crashes. There are several tools to track this kind of error such as NASA Java Pathfinder (JPF) to detect Deadlock, race problem, heap bounds checks, Null Pointer Exceptions and much more advance problem thus finding these problems may take hours, days or months it depends on the size and the complexity of the software.

III. Exixting Techniques for Automated Grading

A. Unit Testing

The main goal of any software testing approach is to check if the software contains errors, produces the right outputs, and follows the specification conducted by the Software tester or the QA team. However, unit testing has reached distinguished prominence in the area of computer science curriculum over the past years [1] and it is one of the most common approaches used nowadays to exam software units or features.

In this test, the targeted software should be clean of any syntax errors, by passing the targeted software to the compiler then apply the test. The output of this test provides the correct or wrong answer based on predetermined inputs derived from the specification document or assignment requirement. Moreover, unit testing is consisting of *test case* and *test methods*; each test case is consisting of one or more test methods that tests a unit or a part of the software code.

In an automated grading system, the instructor responsible for preparing the 'test case' or multiple test cases as a 'test suit' that covers all the aspects that students should include in their assignment. Fig. 1 shows on the right side the multiple JUnit test cases that test a simple Java assignment on the left side written by an introductory student in Rwaq MOOC, each test case assigned to a certain method and has a weight or score.

A comparison between five unit testing tools is represented in Table I.

(Student Assignment File)

```
import java.util.Scanner;
 public class Numbers {

 public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int x, y, z;
        System.out.print("Enter three numbers: ");
        x = s.nextInt();
        y = s.nextInt();
        z = s.nextInt();
        System.out.println("Sum = " + (x + y + z));
```

(Test case written by the instructor)

```
@Test
  public void testNums1() throws Exception
  {
     // first check if the class was loaded. if not issue an
  error.
    if (!numbersFound)
  throw new Exception("class Numbers was not found or
  misspelled.");
    try {
  // prepare for I/O redirection from input/output stream
  to our own
  final ByteArrayOutputStream outContent = new
  ByteArrayOutputStream();
  PrintStream oldStdOut = System.out;

  // CHANGE INPUT VALUE. if program needs two input
  value then
    // separate by space.
    // For example: 1 2 ==> will give the two input values
  1 and 2
  final ByteArrayInputStream in = new
  ByteArrayInputStream(
                                  "1 5 7".getBytes());

  InputStream oldStdIn = System.in;
  System.setOut(new PrintStream(outContent));
  System.setIn(in);

  // load main method
  Method m = getMethod("Numbers", "main",
  String[].class);
  String[] params = null;
  m.invoke(null, (Object) params);

  // check assignment output
  assertTrue("Program output is not correct or
  misspelled. Here is the output:\n" +
  outContent.toString(),
    outContent.toString().matches("(?s).*13.*"));
```

Fig. 1. Example of JUnit Test Cases.

B. Sketching Synthesis and Error Statistical Modeling (ESM)

In [2], the authors introduced a new tool based on sketching synthesis and ESM to provide an instant feedback for introductory programming assignments. The introduced tool was applied on "Introduction to Computer Science and Python Programming Language" that offered by MIT. The key idea behind this method is to provide the system with a reference implementation (best answer) for a simple computational problem such as 'compute derivatives'. Fig. 2 is an example of a reference that is used as the specification for student submissions.

TABLE I. UNIT TESTING TOOLS

Testing Tools	License	Reporting	Configuration/Setup	Error detection		
				Runtime	Syntax	Logical
Junit4	Open source	Clear and easy to read and parse	Include .jar file in test directory	N	N	Y
Parasoft JTest	Enterprise	HTML reports And charts	GUI-based unit testing tool	Y	N	Y
TestNG	Open source	Not easy to parse and simplify for student	XML configurations	N	N	Y
Powermock	Open source	Need someone familiar with the tool to read errors	extends other mock libraries such as EasyMock	N	N	Y
JWalk	**free	Not easy to parse and simplify for student	GUI-based unit testing tool	N	N	Y

** *Only* for research or evaluation purposes

```
1 def computeDeriv_list_int(poly_list_int):
2     result = []
3     for i in range(len(poly_list_int)):
4         result += [i * poly_list_int[i]]
5     if len(poly_list_int) == 1:
6         return result      # return [0]
7     else:
8         return result[1:]  # remove the leading 0
```

Fig. 2. The Reference Implementation for Compute Derivative [2].

Therefore, the tool now shall process and analyze the equivalence of the submitted answers with the reference answer. This approach is using constraint-based synthesis technology [2], [3] to efficiently search over a huge space of programs. Precisely, they use the SKETCH synthesizer that uses the SAT-based algorithm [4] to complete program sketches, so that the students meet the given specification. Moreover, Using SKETCH synthesis system allows writing programs while leaving fragments of it undefined as holes. The synthesizer fills up the contents of these holes such that the program conforms to a specification provided regarding a reference implementation. The synthesizer uses the CEGIS algorithm [5] to compute the values for generated holes and uses bounded symbolic verification techniques for producing equivalence check of student submitted implementation and the reference implementation. Finally, the synthesizer passes the solution to the tool feedback generator to parse the error if found, and translates the output to natural language that students can understand, see Fig. 3.

The generated feedback takes around 40 seconds for each submission and successfully provides feedback on over 64% of wrong answers. The limitations of this tool are as following:

- The tool does not check the structural requirements
- The tool does not accept large constant value.

The tool does not support OOP

C. Peer-To-Peer Feedback

In this approach, the instructor makes the students randomly grade each other's answers. This approach may help students to identify and get used to errors causes, but many problems encountered [6] in systems using this approach such as no instance feedback (students may wait for a long time to get a feedback) and wrong or incomplete feedback due to students limited knowledge especially, introductory students. Finally, students trust and respond to their instructor's comments rather than their peer feedback. In addition, many students find peer feedback hard and not easy to gauge.

D. Random Inputs Test Cases

This approach is proposed in [7], were instructor prepares a set of independent inputs that used to check if the student assignment output is false positive or false negative. However, using this approach to grade students' assignments is very limited and weak since it does not give any feedback that shows the students error; if exist. The objective of this test is to check if the students have determined the correct output or not, so in this case, students will have only two possible grades 0 or 10.

E. Pattern Matching

In pattern matching [6], the instructor provides a specification of the output that a correct assignment will be assumed to generate, and system requests the Unix Lex and Yacc [7] tools (Yet Another Compiler) to create a program that verifies that the output from the student submitted solution meets the provided specification. This technique has many disadvantages since it only accepts and gives a grade to perfect matching solutions. Instructors cannot break down the pattern to distribute the grades on methods.

F. Comparison

The following comparison in Table II shows why Unit testing better than the other techniques.

Three different student submissions for `computeDeriv`

```
1 def computeDeriv(poly):
2     deriv = []
3     zero = 0
4     if (len(poly) == 1):
5         return deriv
6     for e in range(0,len(poly)):
7         if (poly[e] == 0):
8             zero += 1
9         else:
10            deriv.append(poly[e]*e)
11    return deriv
```

(a)

```
1 def computeDeriv(poly):
2     idx = 1
3     deriv = list([])
4     plen = len(poly)
5     while idx <= plen:
6         coeff = poly.pop(1)
7         deriv += [coeff * idx]
8         idx = idx + 1
9         if len(poly) < 2:
10            return deriv
```

(b)

```
1 def computeDeriv(poly):
2     length = int(len(poly)-1)
3     i = length
4     deriv = range(1,length)
5     if len(poly) == 1:
6           deriv = [0]
7     else:
8         while i >= 0:
9             new = poly[i] * i
10            i -= 1
11            deriv[i] = new
12    return deriv
```

(c)

Feedback generated by our Tool

The program requires **3** changes:

- In the return statement **return deriv** in **line 5**, replace **deriv** by **[0]**.
- In the comparison expression **(poly[e] == 0)** in **line 7**, change **(poly[e] == 0)** to **False**.
- In the expression **range(0, len(poly))** in **line 6**, increment **0** by **1**.

(d)

The program requires **1** change:

- In the function **computeDeriv**, add the base case at the top to return **[0]** for **len(poly)=1**.

(e)

The program requires **2** changes:

- In the expression **range(1, length)** in **line 4**, increment **length** by **1**.
- In the comparison expression **(i >= 0)** in **line 8**, change operator **>=** to **!=**.

(f)

Fig. 3. The a,b,c Shows Different Student Submission for the Same Problem and the Feedback for Each Submission Generated by the Feedback Generator [2].

TABLE II. COMPARISON OF TOOLS AND TECHNIQUES USED TO GRADE STUDENTS' JAVA ASSIGNMENTS

Tools and techniques	**Sketching synthesis and error statistical modeling**	**Peer-to-peer feedback**	**Random input test cases**	**Pattern Matching**	**Unit testing**
Execution time	Fast (less than 10 sec in many cases)	40-60 sec in average	Slow, takes hours in many cases	Fast (less than 10 sec in many cases)	Average, it takes 30 sec
Reliability	Accurate (depends on the written test case)	Can detect 64% of students errors	Not reliable it depends on student knowledge	In some case, if all possible inputs are covered	In some case, if all possible outputs are covered
Dependency Test	Supported	Supported	Supported	Not Supported	Not Supported
Instant Feedback	Yes	Yes	NO	Yes	Yes
Support Oop	Yes	No	Yes	No	No

IV. Existing Systems for Automated Grading

The automated grading system of student work has been reviewed for decades. For example, automatic grading of short quizzes involves three types of questions, short answer, multiple-choice and true or false. These questions have been a typical feature of most e-learning system such as Web-Work (an online student homework system for sciences and math courses), Web-CAT (Automatic Grading Student programming assignment), Web-based Grading, and Class-Marker (Online test maker). Grading systems have been interesting in large-scale educational institutions with a substantial number of students. These systems can be classified into three categories [8] (a) Automatic grading systems (the grading and the feedbacks generated by the system) (b) Semi-Automatic grading system (grades generated by the system and feedback produced by human) (c) Manual grading system (the grading and feedbacks produced by human).

A. Automated Grading Systems

In [9], Edwards and Pérez-Quiñones from Virginia Tech have introduced a Web-CAT an extensible and customizable open-source and online automated grading system for students Java and C++ programming assignment. Web-CAT is a state of the art in automated grading system, and many instructors around the world including King Saud University (KSU) have used it with the standard Java-TDD integrated plugin to grade an introductory programming course assignment. The Java-TDD is a combination of jar files that help Web-CAT to manage JUnit test libraries. Web-CAT provides many services to students and instructors, such as assignment submission, automated feedback based on predefined test cases, hints that help to fix errors in the code, and generate grades based on the test case report produced by JUnit-Reporter.

However, while using Web-CAT at Dickinson College, department of mathematics and computer science, [10] in introduction to programming course (COMP131), the instructors frequently observed undesirable difficulties from students' side. The students cannot understand the feedback messages generated by the system regarding their submitted assignments. Most of the errors were general and do not reflect the reason of the actual error. Moreover, the same problem was discovered in Introduction to Java programming language at King Saud University, were 50.6% of the enrolled students were unhappy using the system because the generated feedback report was unclear and does not explain the errors type and also how the system generates their scores. Fig. 4 shows an example of a Web-CAT generated report. The system UI is not usable and most of the submitted assignments fail. Students got zero mark due to having syntax errors such as missing semicolon or missing brackets (Web-CAT does not compile the submissions before running the test cases).

On the other hand, instructors noticed that the system configuration is complex and cannot be accessed remotely from outside the college building since it has been installed on a local server. The students must submit their assignments using the college labs or connect to the system through the local network.

The system proposed by H. Kitaya and U. Inoue is another good example of automated grading system [11]. They provide a java assessment tool that conducts a test by using a regular expression application programming interface (API) to compare the student's java assignments with a reference implementation written by the instructors. The system is a web-based application using many technologies that helps to generate testing frameworks such as Java Servlet and JSP on Apache tomcat, see Fig. 5. Writing a regular expression has major drawbacks in grading systems. It is very restricted to a certain format and output type, and it is not easy to construct an expression pattern. Finally, the system generates only a Japanese feedback messages; this is not applicable in any international course.

B. Semi-Automatic Grading Systems

ASys [12] is a notable recent semi-automatic Java grading system, which focuses on checking the student Java assignment source code by applying three mainly phases: Compilation, Analysis, and Testing respectively. The compilation phase is to check if the source code has any syntax errors, and the output of this phase will be reflected by the programming language compiler. Secondly, if the assignment file passes the first phase the system will enter a crucial phase that is the analysis process that uses a domain specific language (DSL). In this phase, the system authors build an assessment template that consists of two libraries, the Java meta- programming library and a DSL on top of the assessment template. The Java library is composed of 70 methods in an 'Inspectors' class that can be used to examine and handle the source code to check the code properties. Therefore, instructors can check the students Java assignments' programmatically by invoking the DSL that allows the system to load and examine the assignment Java file whether the student has implemented it in a correct way or not.

For example, DSL can detect errors in using inheritance, abstract classes and interface by examining a set of evaluation code. If the evaluation code fails, it returns ZERO and transforms to semi-automatically mode to prompt the instructors with a source code that raises the problem, this is illustrated in Fig. 6.

Assessment system (ASSYST) [13] is an example of a legacy grading system built in 1997 on Linux environment machine. ASSYST uses pattern matching technique and black box testing approach to check and conforms that a student solution meets the specification. Furthermore, the system is not promising since it has many problem reported by the students who have used the system to grade their assignments as it freezes if the system face unknown errors either in student code or in the system core. Years later, David W. Juedes from Ohio University designed a new web based grading system (WBGP) [14] inspired by ASSYST. WBGP is originally written in Perl and Java in 2005 and in 2010 the system was redesigned in TCL/TK [15] scripting language and it works only under Linux and Sun Solaris.

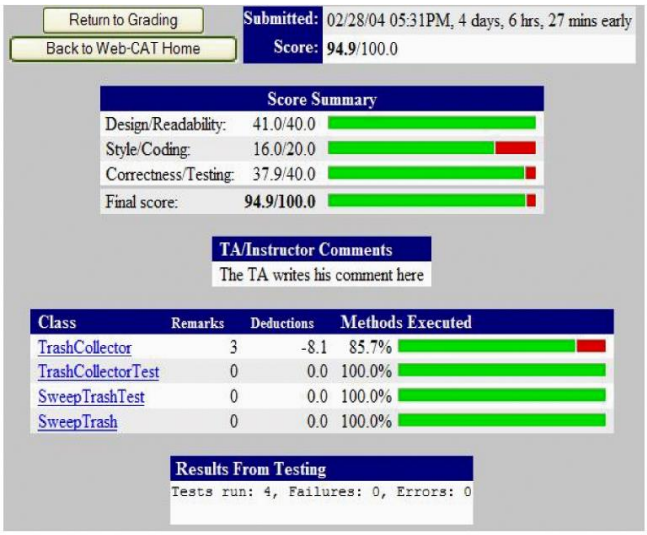

Fig. 4. Web-CAT Generated Report.

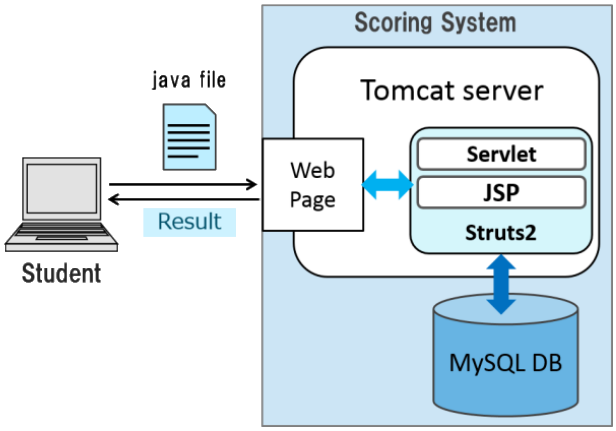

Fig. 5. [11] System Architecture.

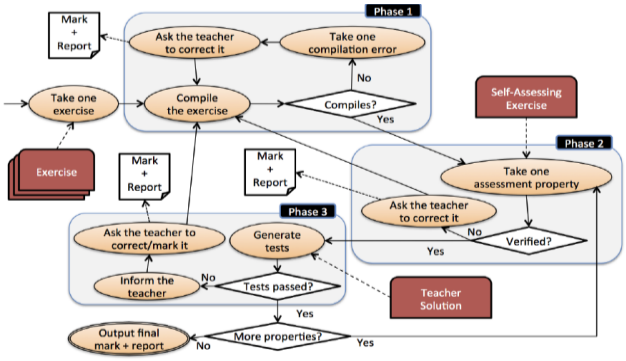

Fig. 6. Asys Data Flow Diagram [12].

The system provides multiple services in addition of grading students' assignments and projects. Services including (i) reporting students' performance and grading progress, (ii) "Measure Of Software Similarity" [16] tool developed by Alex Aiken at UC Berkeley to automatically detect all submitted assignments and projects for evidence of plagiarism.

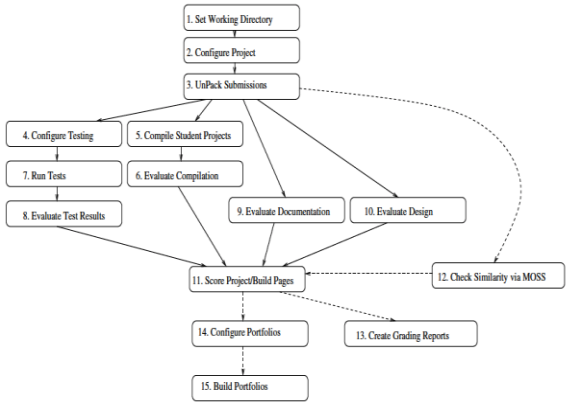

Fig. 7. WBGP Overall Structure [14].

WBGP uses Basic Comment File (BCMT) which provides the grading engine with a list of long and short predefined comments that used to report students through HTML web page of spelling mistakes, syntax errors (reported by the compiler) and output mismatching the random test values. Fig. 7 illustrates the structure of the WBGP system.

While reviewing the system, we discovered many disadvantages such as manual project configuration as shown in Fig. 8. The system requires BCMT files for both, design and implementation, no section/course management, and no support for other Mac OS/Windows operating systems

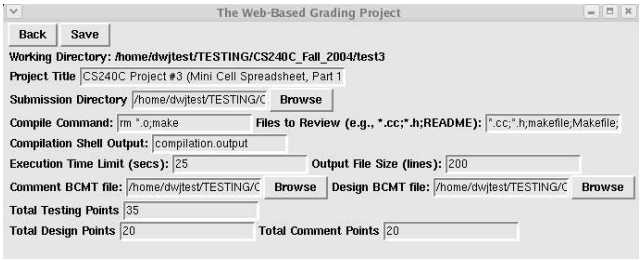

Fig. 8. Project Configuration.

C. Manual Grading Systems

A group of Professors, Teacher Assistants (TAs) and students from University of Toronto, Canada introduced a 'Markus' [17] web-based marking tool to simplify the assignments submission process. It replaces the usual submission approaches (sending emails or submit the assignments in CD/Memory stick, etc.). The main goal of the system is to provide instructors with a simple tool that helps them to give a clear and high- quality feedback to introductory students. The system depends on instructors provided comments or student peer reviews as students can review each other's assignments then instructors can use manual grade and correct the provided reviews/feedback. Markus allows instructors to create a kickoff code (reference implementation) for each assignment to help students solve a certain programming problem.

V. Comparison of Existing Grading System

This section will present in Table III a comparison of all the existed grading systems that have been mentioned in this

article and showing the advantages and disadvantages of using these systems.

TABLE III. COMPARISON OF THE EXISTED GRADING SYSTEMS

	Web-CAT [7]	**WebJavaScroing** [8]	**Markus** [16]	**ASys** [11]	**ASSYST** [13]	**WBGP** [14]
Automated Testing? Automated=1, Semi=0.5, Manuel=0	Automated test suit	Automated, output matching	Manuel, peer review	Automated test cases	Automated test data (Black box testing)	Automated test and pattern matching
Automated Grading? Automated=1, Semi=0.5, Manuel=0	Grades assigned by the system	Grades assigned by the system	Grades assigned by Instructor or TA	Grades assigned by Instructor	Manuel grading by Instructor	Grades assigned by the system
System Usability and Interface USABLE=1, NOT USABLE=0	Not Usable, hard to configure tests and upload assignments	Not usable, no English interface	Usable, Web-based GUI, easy to upload, login, signup	Usable, GUI application, no registration	Not usable, Command base system	Not usable, bad interface design
Sections/Course Support Support=1, Not Supported=0	Supported	Not supported	Supported	Not supported	Not supported	Not supported
Easy to understand system feedbacks EASY=1, MEDIUM=0.5, HARD=0	Medium, as reflected by many students in KSU survey and [8][10]	Hard for non-Japanese actor	Easy, since all approved feedback are generated by Instructors and TA	Easy, feedback generated by Instructor	Hard, feedback generated by the system compiler.	Easy, pre-defined feedbacks by BCMT
Provide hints to correct errors PROVIDE = 1, IN SOMECASE=0.5, NOT PROVIDE=0	Not provided	provided	Provide in some cases (depend on the instructor or TA)	Provide in some cases (depend on the instructor)	Not provided	Provide hints
Main Advantages Over The Other Systems	Integrated with Eclipse and NetBeans IDE	Support multi-submissions	Support group assignments, sections and grade progress chart	precompiled assignments, automatically inform the instructor incase when tests fail.	Measures the code-efficiency by calculating execution time	Plagiarism check
Automated Testing? AUTOMATED=1, SEMI=0.5, MANUEL=0	Automated test suit	Automated, output matching	Manuel, peer review	Automated test cases	Automated test data (Black box testing)	Automated test and pattern matching
Automated Grading? AUTOMATED=1, SEMI=0.5, MANUEL=0	Grades assigned by the system	Grades assigned by the system	Grades assigned by Instructor or TA	Grades assigned by Instructor	Manuel grading by Instructor	Grades assigned by the system
System Usability and Interface USABLE=1, NOT USABLE=0	Not Usable, hard to configure tests and upload assignments	Not usable, no English interface	Usable, Web-based GUI, easy to upload, login, signup	Usable, GUI application, no registration	Not usable, Command base system	Not usable, bad interface design

VI. Conclusion

In this article, we have reviewed many automated grading systems and techniques. A particular attention was paid to the problem of how feedback is generated, to what limit the process is automated, and how much instructor interference needed. Some of the systems were semi-automated, supporting only automated grading or testing. Others are limited to a specific operating system or not appropriate for international courses where there is no proper localization. These limitations are showing a strong need for developing a new technique that fills the gap. As future work, we intend to build a system that provides a fully automated process with the ability to provide consistent grading, precise feedback, and better localization while reducing the time needed by instructor to configure the system.

References

[1] M. Wick, D. Stevenson, and P. Wagner, "Using Testing and JUnit Across the Curriculum," SIGCSE Bull, vol. 37, no. 1, pp. 236–240, Feb. 2005.

[2] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated Feedback Generation for Introductory Programming Assignments," SIGPLAN Not, vol. 48, no. 6, pp. 15–26, Jun. 2013.

[3] M. Ricken and R. Cartwright, "ConcJUnit: Unit Testing for Concurrent Programs," in Proceedings of the 7th International Conference on Principles and Practice of Programming in Java, New York, NY, USA, 2009, pp. 129–132.

[4] A. Solar-Lezama, G. Arnold, L. Tancau, R. Bodik, V. Saraswat, and S. Seshia, "Sketching stencils," 2007, p. 167.

[5] A. Solar-Lezama, R. Rabbah, R. Bodík, and K. Ebcioğlu, "Programming by sketching for bit-streaming programs," ACM SIGPLAN Not., vol. 40, no. 6, p. 281, Jun. 2005.

[6] C. Kulkarni, "Learning design wisdom by augmenting physical studio critique with online self-assessment," Stanford University HCI Group.

[7] D. Juedes, "WBGP (Web-based Grading Project)," School of EECS, Ohiou.

[8] A. Pears, S. Seidman, C. Eney, P. Kinnunen, and L. Malmi, "Constructing a core literature for computing education research," ACM SIGCSE Bull., vol. 37, no. 4, p. 152, Dec. 2005.

[9] S. H. Edwards and M. A. Perez-Quinones, "Web-CAT: Automatically Grading Programming Assignments," SIGCSE Bull, vol. 40, no. 3, pp. 328–328, Jun. 2008.

[10] G. Braught and J. Midkiff, "Tool Design and Student Testing Behavior in an Introductory Java Course," in Proceedings of the 47th ACM Technical Symposium on Computing Science Education, New York, NY, USA, 2016, pp. 449–454.

[11] Y. Akahane, H. Kitaya, and U. Inoue, "Design and Evaluation of Automated Scoring:: Java Programming Assignments," Int. J. Softw. Innov., vol. 3, no. 4, pp. 18–32, Oct. 2015.

[12] D. Insa and J. Silva, "Semi-Automatic Assessment of Unrestrained Java Code: A Library, a DSL, and a Workbench to Assess Exams and Exercises," 2015, pp. 39–44.

[13] D. Jackson and M. Usher, "Grading student programs using ASSYST," ACM SIGCSE Bull., vol. 29, no. 1, pp. 335–339, Mar. 1997.

[14] D. W. Juedes, "Web-based grading: Further experiences and student attitudes," in FRONTIERS IN EDUCATION CONFERENCE, 2005, vol. 35, p. F4E.

[15] B. B. Welch, Practical programming in Tcl & Tk, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2000.

[16] K. W. Bowyer and L. O. Hall, "Experience using 'MOSS' to detect cheating on programming assignments," 1999, vol. 3, p. 13B3/18-13B3/22.

[17] MarkUs: Online marking. University of Toronto, Canada, 2009.