Automated Testing and Marking of Student Programs: Using Web-CAT with Python and Java Assignents

Stefan Brandle

25 February, 2009

Quick History of Automated Marking of Student Programs

- *Earliest I have found*: J. Hollingsworth, "Automatic Graders for Programming Classes", Communications of the ACM, October, 1960. Used punch cards.
- Papers I have found
  - 1960-1970: 3 papers
  - 1970-1980: 1 paper
  - 1980-1990: 11 papers
  - 1990-2000: 28 papers
  - 2000-present: 41+ papers

Reason #8 to Automate Marking

- Time
  - Assume 100 students in the class; 1 marked assignment every two weeks; 5 minutes to process each assignment
  - 100 students/assigmnent * 5 minutes/student * 1 hours/60 minutes = 8.3 hours/assignment (~1 day)
  - 8.3 hours/assignment * 7 assignments/semester * 2 semesters/year * 8 hours/working day = 14.5 working days/year

Reason #7 to Automate Marking

- Consistent Marking of Assignments
  - Inter-rater and intra-rater reliability is difficult
  - Inter-rater: agreement among different people rating (marking) an artifact (document, program, painting, poem, etc.)
  - Intra-rater: agreement by the one person rating the same or an equivalent artifact at different different points in time

Reason #6 to Automate Marking

- Makes it possible for students to rework the assignments and achieve mastery
  - It is demanding for an instructor to mark one submission per student.
  - I have read about a few instructors who tried saying “If you submit your program early, I will mark it and return it to you. Then you can fix the errors and resubmit it before the deadline.”
  - Those instructors only try that policy once!

Reason #5 to Automate Marking

- Makes it possible for students to know their marks right away
  - Students can submit code and be marked immediately at any time, even 3:17am
  - Students are happier
  - Instructor is happier

Reason #4 to Automate Marking

- Makes it reasonable to do continuous assessment
  - Frequent programming assignments are important for continuous assessment
  - Marking those assignments “by hand” discourages instructors from doing continuous assessment
  - Automated marking is a good tool for continuous assessment

Reason #3 to Automate Marking

- Makes it reasonable to assign more complex problems
  - With hand marking, “time-to-grade” can dominate the decision about what to assign
  - Should be based on what is most useful to the students
  - Automated marking essentially eliminates the time-to-grade issue

Reason #2 to Automate Marking

- Makes it easier to teach students to test their own code well
  - With some systems – such as Web-CAT – students can be forced to write and submit their own test suites
  - This can be used even in the first year to teach students superior software development habits

Reason #1 to Automate Marking

- Makes it possible to retain your sanity
  - I have had the privilege of marking assignments for a module with 120 students
  - Afterwards, I was almost willing to find a new job as a garbage collector in order to avoid the marking

http://www.edupics.com/en-coloring-pictures-pages-photo-garbage-collector-i6567.html

What Not to do With Automated Marking

- The Halting Problem
  - "Given a description of a program and a finite input, decide whether the program finishes running or will run forever, given that input."
  - "Alan Turing proved in 1936 that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist."
  - In general, no program – given the source code for other programs – can determine for all other programs whether they will even stop, let alone whether they are "correct".
- **Implication**: do not try to have an automated program read the source for other programs and determine whether they are correct

http://en.wikipedia.org/wiki/Halting_Problem

How Automated Marking is Typically Done

- Approach #1: Black box input/output testing
  - Run the compiled program
  - Feed it input selected carefully so as to test typical cases and boundary cases
  - Compare program output to known correct output for those input cases
  - Run a timer to catch infinite loops
- This is how ACM programming contests verify results

How Automated Marking is Typically Done

- Approach #2: Measure changes in program state
  - Set program state (precondition)
  - Run a particular function
  - Verify that program stated changed correctly (postcondition/results)
  - How unit testing is done

How Automated Marking is Typically Done

- 3: Static analysis (analyze non-running code)
  - Have programs verify program style, internal documentation, etc.
  - Relatively sophisticated free tools available (especially for Java)
- 4: When students write their own unit tests, can do coverage analysis
- 5: Verify correct dynamically allocated memory usage
- 6: Anything else useful that can be automated

The xUnit Testing Approach

- SUnit: Unit testing framework for Smalltalk by "the father of Extreme Programming", Kent Beck.
- xUnit: JUnit, CppUnit, CxxUnit, NUnit, PyUnit, XMLUnit, etc.
- xUnit architecture is an entire talk by itself!

Web-CAT

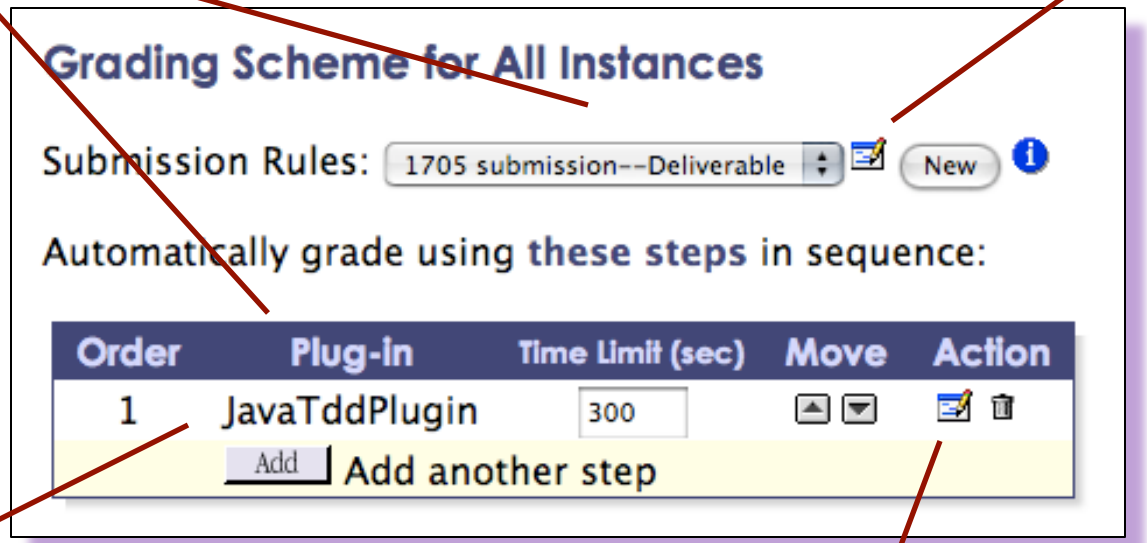
- Dr. Stephen Edwards at Virginia Tech developed Web-CAT to support automated marking of student programs and student-written tests
- Built my own system (Touché Autograder)
- More advantageous for the university community to participate in his better-known, better-funded, and more advanced project
- Web-CAT Premier Award Presentation

Web-CAT: Grade it your way

You decide the balance between automated grading and manual inspection

Use **plug-ins** for a variety of languages, or write your own!

Decide when and how students can submit, including early bonuses and late penalties



Plug-in settings and submission policies can be **reused** over and over

Parameterized plug-ins further extend your options

Web-CAT: Instant results

Students see results in their web browser within minutes

Scoring overview is backed up by detailed line-by-line results in each file

Add overall comments, or write detailed info in-line in source files

Assignment	CS 1705(11689): Program 3: URLHarvester try #19
Name	[redacted] Partners
Submitted	03/25/07 05:07PM, 6 hrs, 47 mins early
Total Score	88.3/100.0

Grading complete? Regrade Submission View Other Submissions

Score Summary

Design/Readability:	34.0	/40.0
Style/Coding:		20.0/20.0
Correctness/Testing:		34.3/40.0
Final score:		88.3/100.0

Position in class: Show Graphs

Done?	File	Remarks	Deductions	Methods Executed
	README.TXT	0	0.0	
✓	URLHarvester.java	2	-4.0	100.0%
✓	URLHarvesterTest.java	2	-2.0	100.0%

Full Printable Report

TA/Instructor Comments

```
<pre>
Commenting/Naming/Style Good          8/10
Required Behavior       Good          8/10
Encapsulation           Good          8/10
Design/Abstraction      Excellent     10/10
Automated Style Checks  Excellent     20/20
Correctness/Testing     Good          34.3/40
-----------------------------------------------
Total                                 88.3/100


GTA: Matthew Thornton
Good job.
```

Web-CAT: Comment on student code

Combine manual code inspection with automated grading results

Leverage industrial-strength tools to run tests, measure code coverage, and check style guidelines

WYSIWYG comment editing right in your browser

Add Comments to This File

Assignment	CS 1705(11689): Program 3: URLHarvester try #19
Name	[redacted]
Submitted	03/25/07 05:07PM, 6 hrs, 47 mins early
Total Score	88.3/100.0
File Name	URLHarvester.java
File Score	-4.0 points lost

Deductions	TA	Tools/Testing		Total
URLHarvester	-4.0	0.0	-4.0	
others	-2.0	-2.0	-4.0	
Total	-6.0	-5.7	-11.7	88.3/100.0

Save & Continue
Save & Finish Later
Save & Mark Done
Cancel

Select from previous comments | Error | To Everyone | B I U

```
102 0                    outStream.println("    " + deep);
103                          }
104                      }
105                  }
106              }
107 2            while (initial != null)
```

Error [Matthew Thornton] : -2.0
A lot of the functionality of this loop could have been put into the previous control structure You only need one primary looping structure in this method and then a loop within that loop.

```
108              {
109 4                initial = getNextHrefURL(in);
110 4                if (initial != null)
111                  {
112 2                    outStream.println(initial);
113 2                    if (initial.substring(0, 4).equals("http"))
114                      {
115 2                        InputStream deepStream = new URL(initial).openStream();
116 2                        Scanner inDeepStream = new Scanner(deepStream);
```

http://web-cat.org

Web-CAT Demonstration

- Python
- Java
- Depending on time, demonstrate PyUnit and JUnit from the command-line

References

- Unit testing: http://en.wikipedia.org/wiki/Unit_testing
- xUnit: http://en.wikipedia.org/wiki/XUnit
- Web-CAT home: http://web-cat.cs.vt.edu/WCWiki/
- "Simple Smalltalk Testing", in Kent Beck's Guide to Better Smalltalk, Donald G. Firesmith Ed. , Cambridge University Press, 1998.
- JUnit: http://junit.org