Tutorial: Automated Grading of Student Programming Assignments

Stefan Brandle

(sbrandle@cse.taylor.edu - 765-998-4685)

Session Outline : Automated Grading

- Introduction
  - Top 8 reasons to automate grading
  - Example of Grader's Nirvana: Web-CAT + turingscraft
  - History of automated grading
- Technology
  - Approaches to automated grading
  - Examples of underlying technology
    - Python
    - C++
    - Java
- Philosophy
  - What cannot be graded automatically
  - What can be graded automatically
  - Pedagogic issues
- Bonus material
  - Web-CAT demo
  - Web application testing with Selenium
- References

Top 8 Reasons to Automate Grading

Reason #8 to Automate Grading

- Time
  - Assume 40 students in the class; 1 graded assignment every two weeks; 5 minutes to process each assignment
  - 40 students/assignment * 5 minutes/student * 1 hours/60 minutes = 3.3 hours/assignment
  - 3.3 hours/assignment * 7 assignments/class * 6 classes/year * 1 day/8 hours
    ~= 17 working days/year
  - Your mileage my vary (to your detriment)

Reason #7 to Automate Grading

- Consistent grading of Assignments
  - Inter-rater: agreement among different people rating (grading) an artifact (document, program, painting, poem, etc.)
  - Intra-rater: agreement by the same person rating the same or an equivalent artifact at different different points in time
  - Want good inter-rater and intra-rater reliability
    - Hard to obtain

Reason #6 to Automate Grading

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

Reason #5 to Automate Grading

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

Reason #4 to Automate Grading

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

Reason #3 to Automate Grading

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

Reason #2 to Automate Grading

- 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 (TDD – Test Driven Development)

Reason #1 to Automate Grading

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

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

Examples of Grader's Nirvana:
Web-CAT
Turing's Craft (talk to them afterwards)

Web-CAT

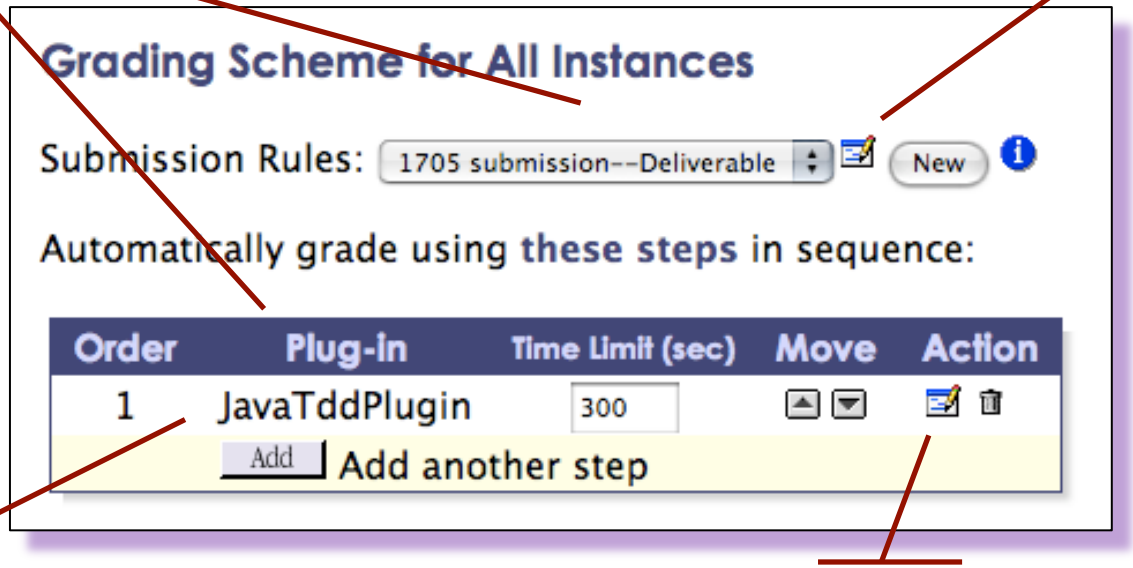
- Stephen Edwards at Virginia Tech developed Web-CAT to support automated grading of student programs and student-written tests (TDD)
- Built my own system (Touché Autograder)
- I decided that it was better for the overall community if I participated in his better-known, better-funded, and more advanced project

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

http://web-cat.org

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	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

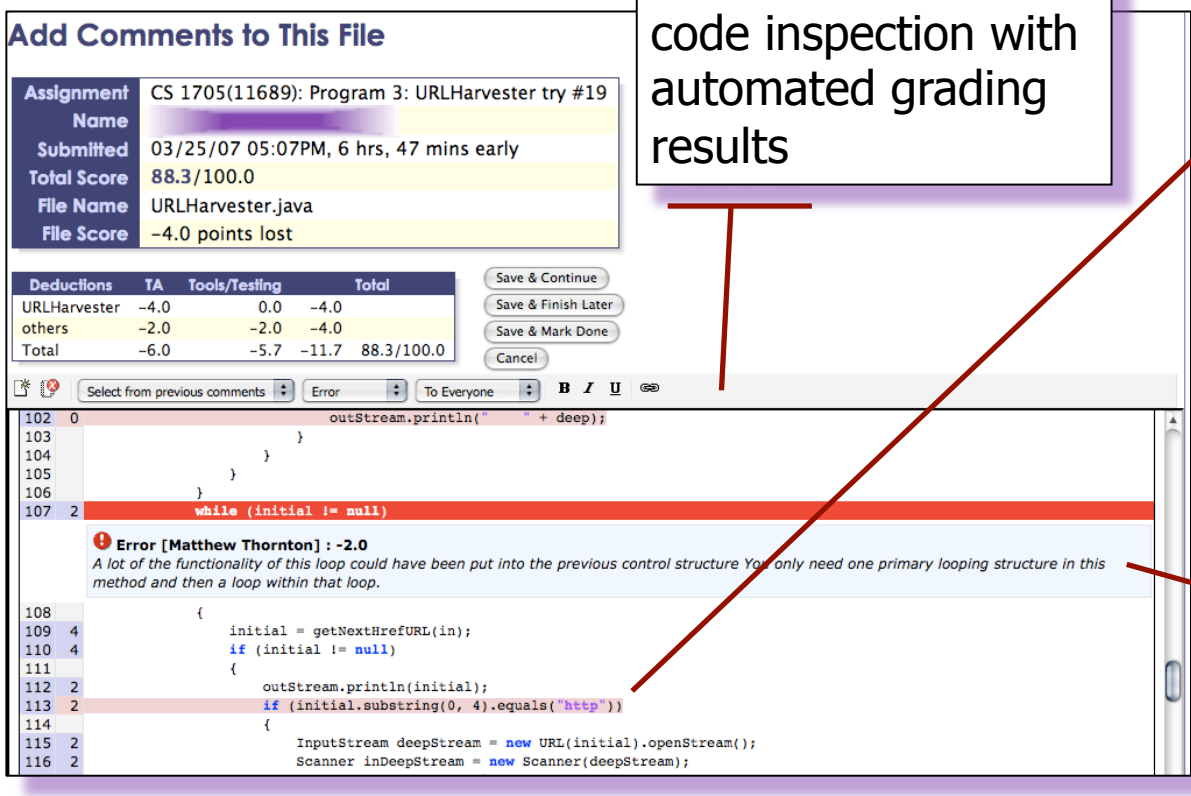
TA/Instructor Comments

```
<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.
```

http://web-cat.org

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

http://web-cat.org

History of Automated Grading

A Quick History of Automated Grading 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 at last count

Approaches to Automated Grading

How Automated Grading 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
  - Deal with problems like infinite loops and too much output by running in special "containers" with timers, I/O limitations, and more.

- Black box input/output testing is how programming contests typically verify results

How Automated Grading is Typically Done

- Approach #2: Measure changes in program state
  - Set program state (precondition)
  - Run student's snippet of code/function/set of functions
  - Verify that program state changed correctly (postcondition/results)
  - Unit testing is done this way

How Automated Grading 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

Brief Reminder from Your Sponsor: Just Because You Can, It Doesn't Mean …

- Presenting the technology here
- Don't become entry in SIGCSE "It seemed like a good idea at the time"?
- Automated assessment is ONE available tool
- Big picture
  - Much more than automated grading
  - Whole software development philosophy

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!

Unit Testing

- Unit testing: a method of testing that verifies the individual units of source code are working properly. (en.wikipedia.org/wiki/Unit_testing)
- Unit testing: The testing done to show whether a unit (the smallest piece of software that can be independently compiled or assembled, loaded, and tested) satisfies its functional specification or its implemented structure matches the intended design structure. (testinghelp.googlepages.com/QAglossaryofterms.doc)
- What software can unit testing be done on?

Unit Testing

- Frequent features of unit tests
  - Name test functions test*FunctionName*
  - Any function named test* is automatically run
  - Results reported by a “test runner”
  - Setup
  - Teardown

xUnit Architecture

- Test case – the base class
- Test suite – a class for aggregating unit tests
- Test runner
  - Reports test result details
  - Simplifies the test
- Test fixture
  - Test environment used by multiple tests
  - Provides a shared environment (with setup, tear-down, and common variables) for each test
- A set of assertion functions
  - E.g., assert(expression, “string to print if false”)

Other Unit Test Terms

- Stubs – “the smallest amount of code that can fail”
  - Make a function with just enough code to compile
  - Doesn’t actually meet the requirements
  - Useful for setting up the test suite
  - Generating this is part of TDD (Test Driven Development) philosphy
- Mock or fake objects
  - Used to simulate (transparently, if possible) some other object
  - Could simulate another class, a database, a network connection
- Test harnesses
  - The testing environment within which a units are tested
- Regression testing
  - Testing to ensure that previously working units still work
- Test coverage
  - What percentage of all code to be tested is actually tested (covered)

Examples of Automated Grading Tools

- Python
  - Unit testing: PyUnit
  - Black box I/O (Web-CAT)
- Java
  - Unit testing: JUnit within eclipse
- C++
  - Unit testing: CxxTest
  - Black box I/O (Web-CAT)
- Web sites
  - Unit testing: Selenium

Testing Java Code

```
// Simple one-file point class
class Point {
   int x = 0;
   int y = 0;

   Point( int xCoord, int yCoord ) {
     this.x = xCoord;    // Note use of “this”  ☺
     this.y = yCoord;
   }

   void set( int xCoord, int yCoord ) {
     this.x = xCoord;
     this.y = yCoord;
   }

   void move( int xDelta, int yDelta ) {
     this.x += xDelta;
     // Deliberate error in changing y. Mimicks copy-n-paste error.
     // Activate one of the two lines.
     this.y += xDelta;
     //this.y += yDelta;
   }
}
```

```
//Test class for the Point class.

import junit.framework.*;

public class PointTest extends TestCase
{
   // Creates a new Point object at (0,0)
   public void setUp() {
      point = new Point(1,2);
   }

   //  Public Methods
   public void testInitial() {
      assertEquals( point.x, 1 );
      assertEquals( point.y, 2 );
   }

   public void testSet() {
      point.set( 3, 1 );
      assertEquals( point.x, 3 );
      assertEquals( point.y, 1 );
   }
```

```
   public void testMove() {
      point.move( 7, 2 );
      assertEquals( point.x, 8 );
      assertEquals( point.y, 4 );
   }

   private Point point;

   // Unit Testing main function. Used to
   // run the unit tests from the
   // command line. Type "java PointTest”
   // to start the tests (if junit is in
   // CLASSPATH).

   public static void main(String args[]) {
      org.junit.runner.JUnitCore.main(
                                    "PointTest");
   }

}
```

Testing Python Code

```python
#!/usr/bin/env python
# This is a trivial example of a one-file assignment
"""Simple one-file point class"""

class Point:
  x = 0
  y = 0

  def __init__(self, xCoord, yCoord):
    self.x = xCoord
    self.y = yCoord

  def set(self, xCoord, yCoord):
    self.x = xCoord;
    self.y = yCoord;

  def move(self, xDelta, yDelta):
    self.x = self.x + xDelta;
    # Deliberate error in changing y. Mimicks copy-n-paste error.
    # Activate one of the two lines.
    self.y = self.y + xDelta;
    #self.y = self.y + yDelta;
```

```
import point
import unittest

class PointTests(unittest.TestCase):

  def setUp(self):
    self.point = point.Point( 1, 1 );

  def testCreatePoint(self):
    """Test point creation"""
    self.assertEqual( 1, self.point.x,
          "x attribute not correctly set" )
    self.assertEqual( 1, self.point.y,
          "y attribute not correctly set" )

  def testSetPoint(self):
    """Test setting point attribute"""
    self.point.set( 11, 7 )
    self.assertEqual( 11, self.point.x,
         "x value setting incorrectly done" )
    self.assertEqual( 7, self.point.y,
         "y value setting incorrectly done" )
```

```
def testMovePoint1(self):
    """Test point creation"""
    self.point.move( 5, 3 )
    self.assertEqual( 6, self.point.x,
          "x change not correctly done" )
    self.assertEqual( 4, self.point.y,
          "y change not correctly done" )

  def testMovePoint2(self):
    """Test point creation"""
    self.point.move(0,0)
    self.assertEqual( 1, self.point.x,
          "x change not correctly done" )
    self.assertEqual( 1, self.point.y,
          "y change not correctly done" )

if __name__ == '__main__':
  unittest.main()
```

Testing C++ Code

```
// Point.h
// Simple one-file point class

class Point {
  public:
    int x;
    int y;

    Point( int xCoord, int yCoord );
    void set( int xCoord, int yCoord );
    void move( int xDelta, int yDelta );
};
```

```
// Point.cpp
#include "Point.h"

Point::Point( int xCoord, int yCoord ) {
    this->x = xCoord;
    this->y = yCoord;
}

void Point::set( int xCoord, int yCoord ) {
    this->x = xCoord;
    this->y = yCoord;
}
void Point::move( int xDelta, int yDelta ) {
    this->x += xDelta;
    // Deliberate error in changing y.
    // Mimicks copy-n-paste error.
    // Activate one of the two lines.
    //this->x += xDelta;
    this->y += yDelta;
}
```

```
/**
 * Test class for the Point class.
 */

#ifndef POINTTEST_H
#define POINTTEST_H

#include <cxxtest/TestSuite.h>
#include "Point.h"

class PointTest : public CxxTest::TestSuite
{
  public:
    void setUp() {
        point = new Point(1,2);
    }

    void tearDown() {
        delete point;
    }

 void testInitial() {
        TS_ASSERT_EQUALS( point->x, 1 );
        TS_ASSERT_EQUALS( point->y, 2 );
    }

    void testSet() {
        point->set( 3, 1 );
        TS_ASSERT_EQUALS( point->x, 3 );
        TS_ASSERT_EQUALS( point->y, 1 );
    }

    void testMove() {
        point->move( 7, 2 );
        TS_ASSERT_EQUALS( point->x, 8 );
        TS_ASSERT_EQUALS( point->y, 4 );
    }

 private:
    Point* point;
};

#endif
```

Philosophy

- What cannot be done
- What can be done
- Pedagogic issues

What Cannot Be Automated Graded

- The Halting Problem
  - Unless in mood for a big CS award, don't take on 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 whether they are "correct".

- **Implication**: In general, do not try to have an automated program read the source for other programs and determine whether they are correct.
- *Exception: Can do this for very small pieces of code, but hard to do right. See TuringsCraft.com*

- Grading good design

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

What Can be Automatically Graded?

- Pretty much anything not in the "Cannot be graded automatically"
- Functionality, coding style, memory usage, documentation, …, anything for which you can find a tool that measures it
- Caution:
  Remember "It seemed like a good idea at the time …"?
  - Some things are not a good idea, although they will appear to be good at the time. ☺

Some Pedagogic Issues

- What it means when …
  - Students submit non-compiling code
  - Success is [only] passing the tests
- How many tests to write
  - N test functions for N tests of one function
  - One test function for all N tests
  - Grade can be quite different
- What types of hints to issue
  - Can go from very detailed, to no details
- Improving student behavior/habits
  - Reduce feedback quantity/quality as approach submission deadline
  - Limit number of submissions?
- Teaching students TDD mindset, vs. just assessing their code

Bonus Material

Web-CAT Demonstration

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

Testing Web Applications

Testing Web Applications

- Why test?
  - We should be able to skip this, you know the answer
- What to test?
  - This is harder
- How to test
  - This is perhaps hardest
- One possible answer is Selenium …

Selenium Demonstration

- Demonstration of Selenium running in Firefox
- Project site
  - Main seleniumhq.org
  - Documentation seleniumhq.org/projects/core/reference.html

Selenium Commands

- Actions
  - Commands that manipulate the state of the application
  - E.g. "click this link" and "select that option”
- Accessors
  - Examine the state of the application and store the results in variables
  - E.g. "storeTitle”
- Assertions
  - Like Accessors, but verify that the state of the application is as expected.
  - E.g. "make sure the page title is X" and "verify that this checkbox is checked".

http://seleniumhq.org/projects/core/reference.html

Selenium Commands

- All Selenium Assertions can be used in 3 modes
  - E.g., you can "assertText", "verifyText" and "waitForText"
  - Assert
    - When an "assert" fails, the test is aborted
  - Verify
    - When a "verify" fails, the test will continue execution, logging the failure.
  - WaitFor
    - Wait for some condition to become true (which can be useful for testing Ajax applications).
    - Fail and halt the test if the condition does not become true within the current timeout setting

http://seleniumhq.org/projects/core/reference.html

Other Selenium Concepts

- Element Locators
  - Tell Selenium which HTML element a command refers to
  - E.g., "elementId" and "document.forms [0].element"
- Patterns
  - Supports various types of pattern, including regular-expressions
  - Such as to specify the expected value of an input field, or identify a select option
  - E.g., “*@uom.ac.mu”, “*success*”

http://seleniumhq.org/projects/core/reference.html

Selenium: SetUp/TearDown

- "There are no setUp and tearDown commands in Selenese, but there is a way to handle these common testing operations. On the site being tested, create URLs for setUp and tearDown. Then, when the test runner opens these URLs, the server can do whatever setUp or tearDown is necessary."

Setup and Teardown		
open	/setUpFT	
storeValue	userid	
open	/login	
type	userID	${userid}
click	submit	
open	/tearDownFT?userid=${userid}	

http://seleniumhq.org/projects/core/usage.html

More About Selenium

- http://seleniumhq.org
- Generated Documentation
  - JavaDoc for Selenium Remote Control driver
  - NDoc reference for .NET driver
  - PHPDocumentor for the PHP driver
  - PyDoc reference for the Python driver
  - RDoc reference for the Ruby driver

http://seleniumhq.org/documentation/

References (1)

General

- Unit testing: http://en.wikipedia.org/wiki/Unit_testing
- xUnit: http://en.wikipedia.org/wiki/XUnit
- "Simple Smalltalk Testing", in Kent Beck's Guide to Better Smalltalk, Donald G. Firesmith Ed. , Cambridge University Press, 1998.

References (2)

Unit-Testing Frameworks

- PyUnit: pyunit.sourceforge.net
  pyunit.sourceforge.net/pyunit.html
- xUnit: http://en.wikipedia.org/wiki/XUnit
- JUnit: http://junit.org
- CxxTest: http://cxxtest.tigris.org/
- Selenium: http://seleniumhq.org/

References (3)

Sample automated grading systems

- Web-CAT: web-cat.cs.vt.edu/WCWiki/
- Code Lab®: www.turingscraft.com
- GOAL (Pearson): www.pearsonhighered.com/educator/product/GOAL-Where-virtual-office-hours-are-247/9780136037743.page

Questions?

- Copy of this presentation

  cse.taylor.edu/~sbrandle

- Email: sbrandle@cse.taylor.edu