WEB-CAT – AUTOMATED GRADING TOOL WITH AN INTERACTIVE ENVIRONMENT FOR LEARNING PROGRAMMING

Ann Molly Paul

Section 1

Overview

Need for Automated Grading

Time Management

- Assume 40 students → 2 assignments / week →10 minutes to grade an assignment.
  - (40*2*10)/60 hours
- This amounts to 13 hours a week on grading for an average person

Avoiding Inconsistency

- Inconsistency while grading different code for the same test cases
  - Varying styles of coding
  - Different use of methods
  - Different complexity of methods to do similar operations
  - Challenge for the grader to grade impartially

Opportunity for improvement

- Less time for grading→ more opportunity for students to improve code.
  - It is demanding for an instructor to grade even one submission per student leaving aside option for resubmission
  - Allows students to improve code after an early submission

Speedy Grading

- Makes it possible for students to know their grades right away
  - Students are happier
  - Instructor is happier

Encourages more learning

- Continuous Assessment
  - Less difficulty in grading encourages instructors to give more assignments
  - Improves students programming skills while they try solving different questions

Challenge students

- Makes it reasonable to assign more complex problems
  - Time taken to grade dominates the decision while assigning questions(easy preferred over hard)
  - Automatic grading makes it easier for profs to grade complex problems more accurately

Test driven coding

- Encourages students to code with test cases in mind
  - Web-CAT allows students to write their own test cases.
  - Teaches students Test driven development(TDD)
  - Gives them deeper understanding on the assignment

Section 2

Methodology

Approaches to Automate Grading

Method 1- Black box input/output testing

- Run the compiled program
- Feed it input -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.

- **USES:** In Programming Contests to verify results

Method 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 (post condition/results)
- Unit testing is done this way

Method 3: Static analysis (analyze non-running code)

- Features:
  - Have programs verify program style, internal documentation, etc.
  - Relatively sophisticated free tools available (especially for Java)
- When students write their own unit tests, can do coverage analysis
- Verify correct dynamically allocated memory usage

Section 3

Testing

Unit Testing

- **Definition:** a method of testing that verifies the individual units of source code are working properly
- Shows whether a unit (the smallest piece of software that can be independently compiled or assembled, loaded, and tested) satisfies its functional specification
- Checks if its implemented structure matches the intended design structure.

The xUnit Testing Approach

- This approach is modifies unit testing to test code in different languages and has environments specific to a single language.
- xUnit: JUnit, CppUnit, CxxUnit, NUnit, PyUnit, XMLUnit, etc.

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”)

Section 4

Prior Approaches to Automate Code Evaluation

Curator – Tool to grade programs

- Curator compiles the student program.
- Runs a test data generator to create input for grading.
- Uses a reference implementation as expected output
- Grades by comparing against the reference implementation's output.
- Student receives feedback
- It includes the input used, the student's output, and the instructor's expected output for reference.

Limitations

- Focus on output correctness
  - Score of zero →submissions that do not compile, do not produce output, or do not terminate.
  - Don’t consider →design, commenting, appropriate use of abstraction, testing one's own code, etc.)
- Students are not encouraged or rewarded for performing testing on their own.
- Never perform serious testing of their own programs

Section 5

Web-CAT – In detail

Web-CAT

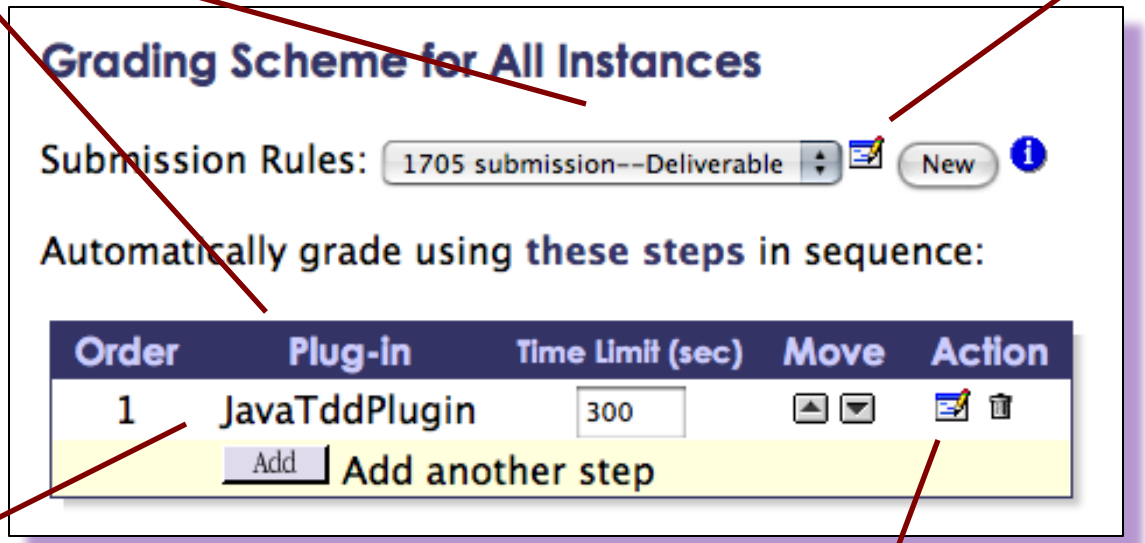
- Stephen Edwards at Virginia Tech developed Web-CAT
- Aim: To support automated grading of student programs.
- Used to grade student-written tests
- Inculcates test driven development (TDD)

GRADING SCHEME

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

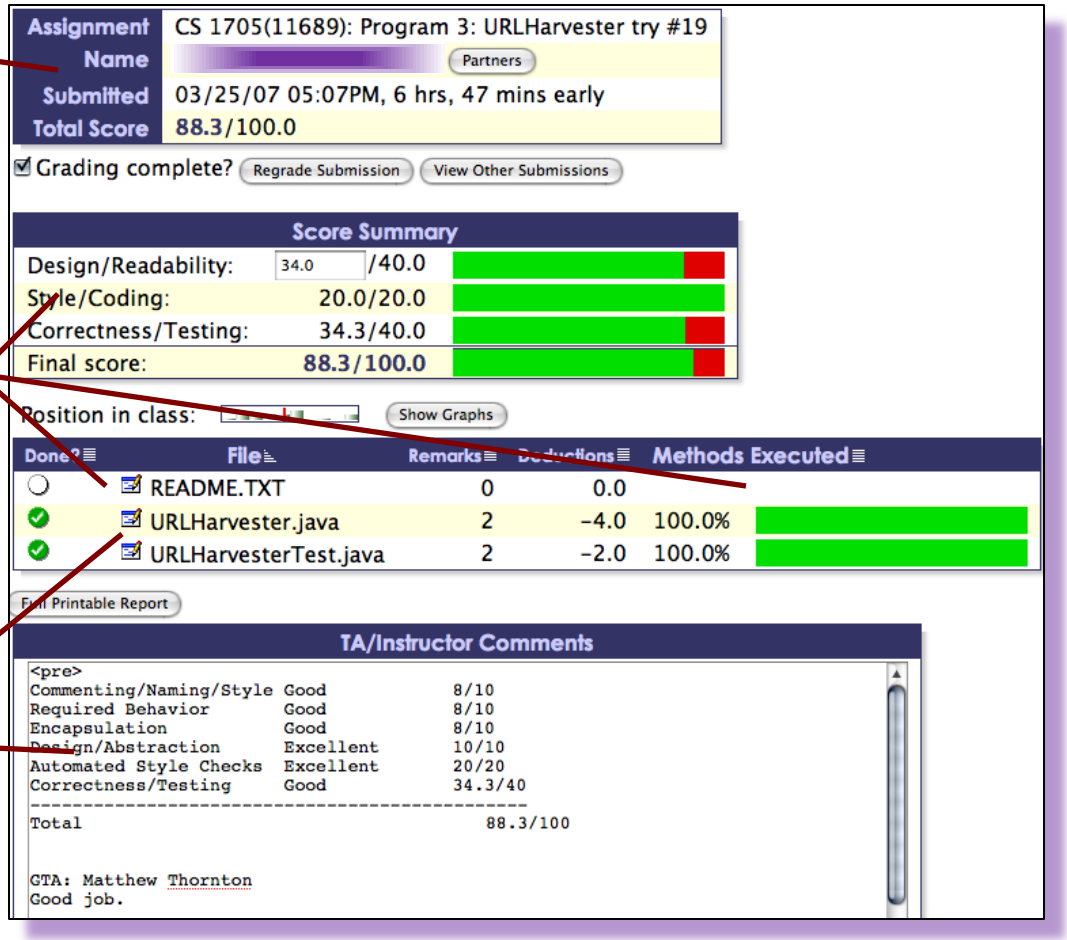
Parameterized plug-ins further extend your options

DISPLAY OF RESULTS – INSTANT!

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



View Assignment Results

Back to Web-CAT Home

Score Summary	
Possible points:	50.0
Deductions:	-7.0
Early bonus:	0.0
Late penalty:	0.0
Final score:	**43.0**

Submission Details	
Project name:	p1-QuadTree
Submission no.:	9
File size:	14840
Submission time:	01/30/03 01:26PM
Deadline:	02/12/03 10:10AM
Late deadline:	02/14/03 10:10AM
Early bonus:	none
Late penalty:	20 points per 1 day late

Correctness Based on Your Tests

Your Program 93% **37 of 40** tests passed

Thoroughness of Your Testing

Your Test Cases 92% **92%** coverage, **40 of 40** tests valid

Score = (93% x 100% x 92%) x 50 = 43

Program Correctness (Your Solution)

```
tddpas.pl v1.2: Testing your submission using p1tests.txt

.....F
case 6 FAILED: empty string on the left
......F
case 13 FAILED: two empty strings
.....................F
case 34 FAILED: merging two overlapping trees
......

Tests Run: 40, Errors: 0, Failures: 3 (92.5%)
```

Test Validity (Reference Solution)

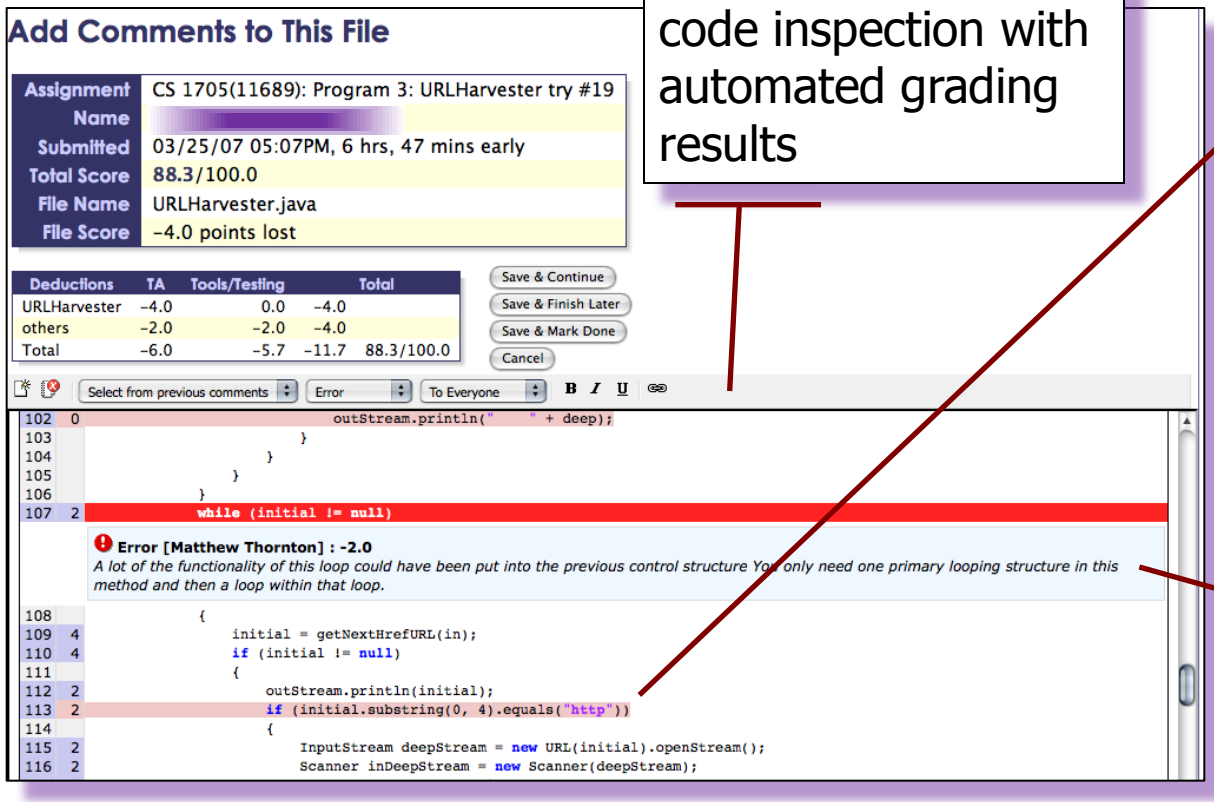
```
tddpas.pl v1.2: Testing reference implementation using p1tests.txt

........................................

Tests Run: 40, Errors: 0, Failures: 0 (100.0%)
```

Done

COMMENTS AND REVIEW



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

Section 6

Contribution to Learning Experience

Road Blocks to learn efficient coding skills

- Student mostly use ‘trial and error’ technique to write code.
- Software testing requires experience at programming- New students are not ready for it.
- Instructors just don’t have the to teach a new topic like software testing
- Course staff already has its hands full assessing program correctness
- Students are concerned about the output and not how to develop the solution

Benefits of Web-CAT

- Easier for students to understand and relate to than more traditional testing approaches.
- Promotes incremental development
- Promotes early detection of errors in code
- Increases the student's understanding of the assignment requirements, by forcing them to explore the gray areas in order to completely test their own solution.

Section 7

Philosophy

3 Aspects

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

What Cannot Be Automated Graded

- The Halting Problem
  - Given a description of a program and a finite input, decide whether the program **finishes running** or will run forever
  - General algorithm to solve the halting problem for all possible program-input pairs cannot exist.(Alan Turing)

Contd.

- Cannot have an automated system read the source code for programs and determine whether they are correct.
  - **Exception:** Can do this for very small pieces of code, but hard to do right
- Design cannot be graded- good/bad

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

Some Pedagogic Issues

- 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

Additional Resources

- Web-CAT: web-cat.cs.vt.edu/WCWiki/
- Code Lab®: www.turingscraft.com

?

QUESTIONS OR COMMENTS