Using Web-CAT to Grade Students on How Well they Test Their Own Code

Stephen Edwards
edwards@cs.vt.edu

Virginia Tech
Department of Computer Science

http://web-cat.org/

NSF DUE-0633594 and DUE-0618663

What is Web-CAT?

- A plug-in-based web application
- Supports electronic submission and automated grading of programming assignments
- Fully customizable, scriptable grading actions and feedback generation
- Lots of support for grading students based on how well they test their own code

Who uses Web-CAT?

- At 38 institutions and growing
- Approaching 10,000 users worldwide
- Since 2003, Virginia Tech's server alone has processed approximately:
  - 264,818 program submissions
  - By 4,135 students
  - In 186 course sections

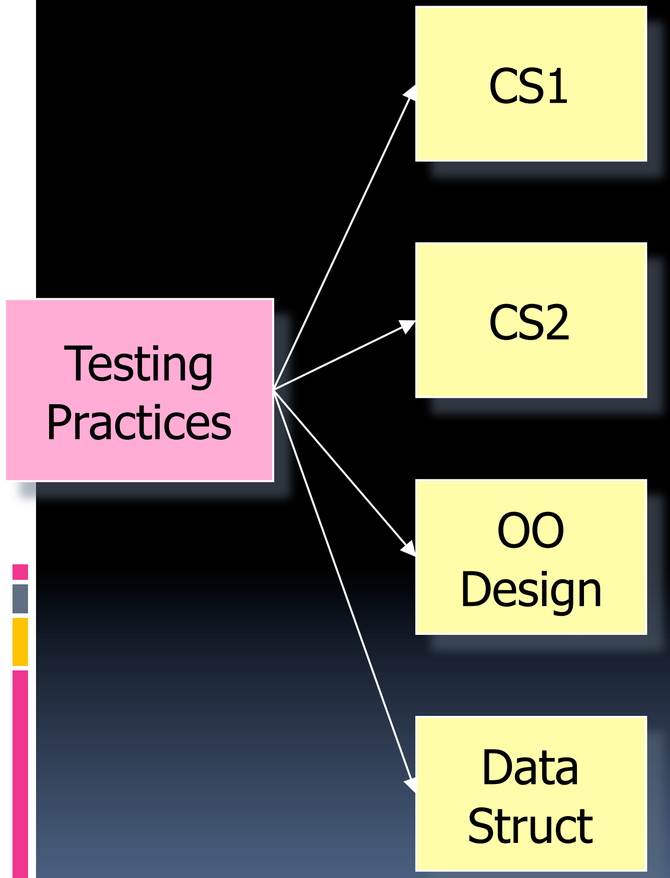
Some shameless plugs …

- In addition to Web-CAT itself, our research group has a number of other testing-related tools available, including:
  - CxxTest (with plug-ins for Eclipse and Visual Studio)
  - Dereferee
  - Electronic submission plug-ins for Eclipse and Visual Studio

More educators are adding software testing to their programming courses

- Now it’s almost routine
- Tools like **JUnit**, and XUnit frameworks for other languages, make it much easier
- Built-in support by many mainstream and educational IDEs makes it much easier
- Many instructors have also experimented with automated grading based on such testing frameworks
- Here are **our experiences** in teaching test-driven development with the help of an automated grader over the past 3 years

Why have we added software testing across our programming core?



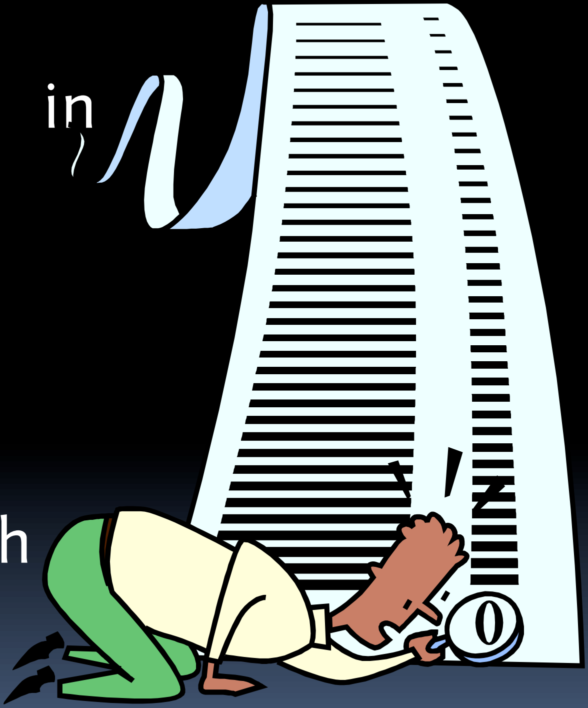
- Students **cannot test** their own code
- Want a **culture shift** in student behavior
- A single upper-division course would have **little impact** on practices in other classes
- So: Systematically incorporate testing practices across many courses

Software testing helps students frame and carry out experiments

- The **problem**: too much focus on synthesis and analysis too early in teaching CS
- Need to be able to read and comprehend source code
- Envision how a change in the code will result in a change in the behavior
- Need explicit, continually reinforced practice in **hypothesizing** about program behavior and then **experimentally verifying** their hypotheses

Expect students to apply testing skills all the time

- Expect students to **test their own work**
- **Empower** students by engaging them in the process of assessing their own programs
- **Require** students to demonstrate the correctness of their own work through testing
- Do this consistently **across many courses**



Test-driven development is very accessible for students

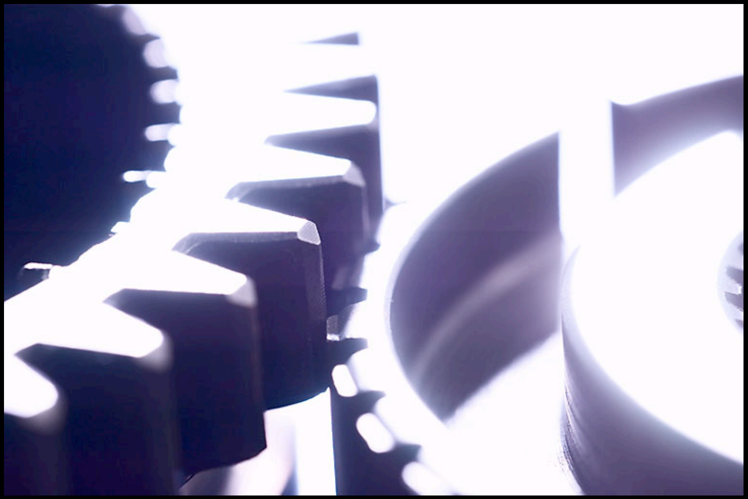
- Also called “test-first coding”
- Focuses on thorough unit testing at the level of individual methods/functions
- “Write a little test, write a little code”
- Tests come first, and describe what is expected, then followed by code, which must be revised until all tests pass
- Encourages lots of small (even tiny) iterations

Students can apply TDD and get immediate, useful benefits

- Conceptually, easy for students to understand and relate to
- **Increases confidence** in code
- **Increases understanding** of requirements
- Preempts “big bang” integration

We use Web-CAT to automatically process student submissions and check their work

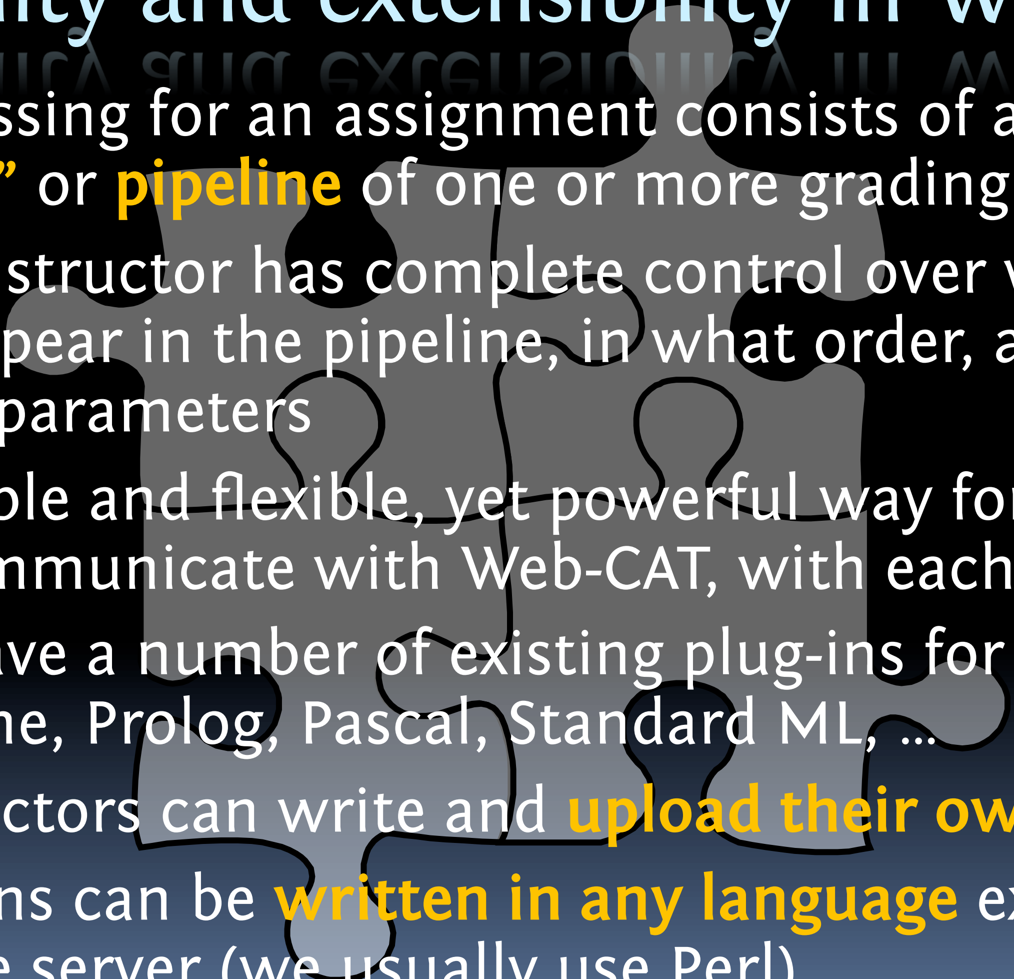
- Web application written in 100% pure Java
- Deployed as a servlet
- Built on Apple's WebObjects
- Uses a large-grained plug-in architecture internally, providing for easily extensible data model, UI, and processing features



Web-CAT's strengths are targeted at broader use

- **Security:** mini-plug-ins for different authentication schemes, global user permissions, and per-course role-based permissions
- **Portability:** 100% pure Java servlet for Web-CAT engine
- **Extensibility:** Completely language-neutral, process-agnostic approach to grading, via site-wide or instructor-specific grading plug-ins
- **Manual grading:** HTML "web printouts" of student submissions can be directly marked up by course staff to provide feedback

Grading plug-ins are the key to process flexibility and extensibility in Web-CAT



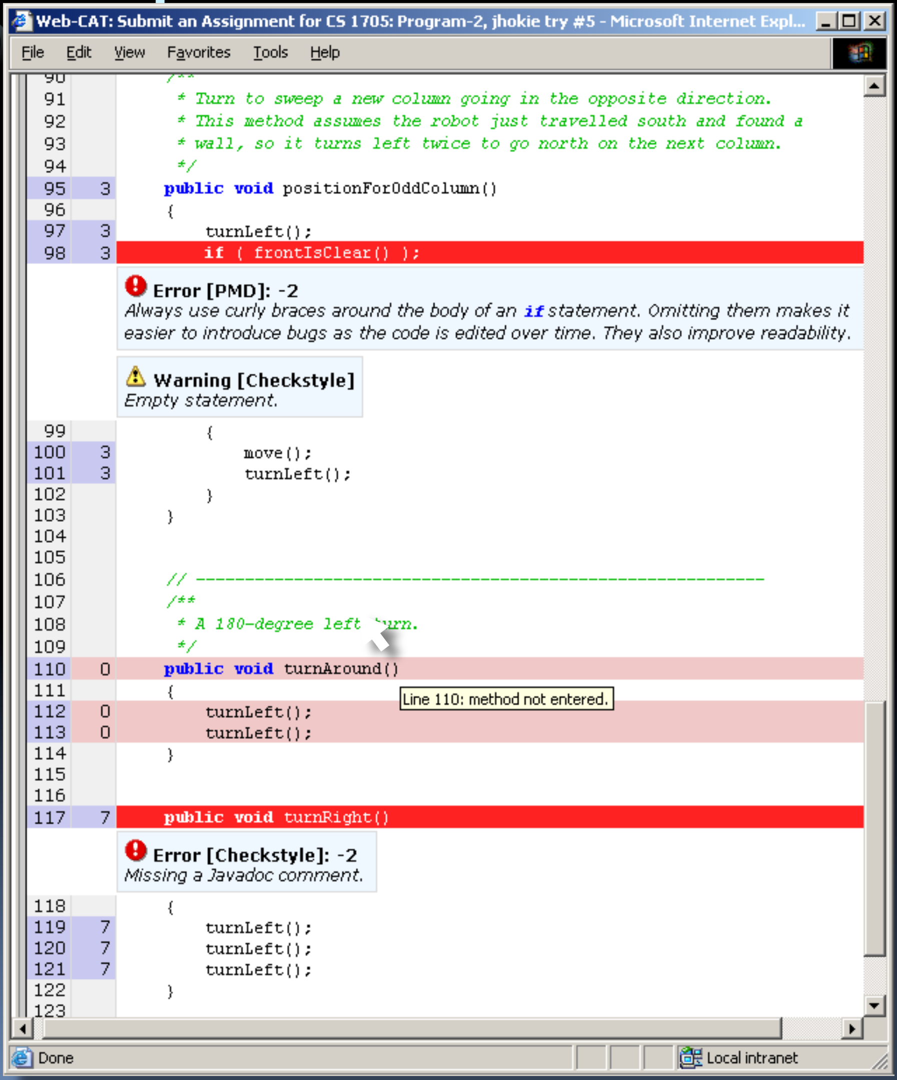
- Processing for an assignment consists of a **"tool chain"** or **pipeline** of one or more grading plug-ins
- The instructor has complete control over which plug-ins appear in the pipeline, in what order, and with what parameters
- A simple and flexible, yet powerful way for plug-ins to communicate with Web-CAT, with each other
- We have a number of existing plug-ins for Java, C++, Scheme, Prolog, Pascal, Standard ML, …
- Instructors can write and **upload their own** plug-ins
- Plug-ins can be **written in any language** executable on the server (we usually use Perl)

The best-known plug-in grades Java assignments that include student tests

- **ANT**-based build of arbitrary Java projects
- **PMD** and **Checkstyle** static analysis
- ANT-based execution of student-written JUnit tests
- Carefully designed Java **security policy**
- **Clover** test coverage instrumentation
- ANT-based execution of optional instructor reference tests
- Unified HTML web printout
- **Highly configurable** (PMD rules, Checkstyle rules, supplemental jar files, supplemental data files, java security policy, point deductions, and lots more)

Web-CAT provides timely, constructive feedback on how to improve

- Indicates where code can be improved
- Indicates which parts were not tested well enough
- Provides as many “revise/ resubmit” cycles as possible


Web-CAT: Submit an Assignment for CS 1705: Program-2, jhokie try #5 - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Done
Local intranet

```
 90          /**
 91           * Turn to sweep a new column going in the opposite direction.
 92           * This method assumes the robot just travelled south and found a
 93           * wall, so it turns left twice to go north on the next column.
 94           */
 95    3     public void positionForOddColumn()
 96          {
 97    3         turnLeft();
 98    3         if ( frontIsClear() );
```

Error [PMD]: -2
*Always use curly braces around the body of an **if** statement. Omitting them makes it easier to introduce bugs as the code is edited over time. They also improve readability.*

Warning [Checkstyle]
Empty statement.

```
 99              {
100    3             move();
101    3             turnLeft();
102              }
103          }
104
105
106          // -----------------------------------------------------------
107          /**
108           * A 180-degree left turn.
109           */
110    0     public void turnAround()
111          {
112    0         turnLeft();
113    0         turnLeft();
114          }
115
116
117    7     public void turnRight()
```

Line 110: method not entered.

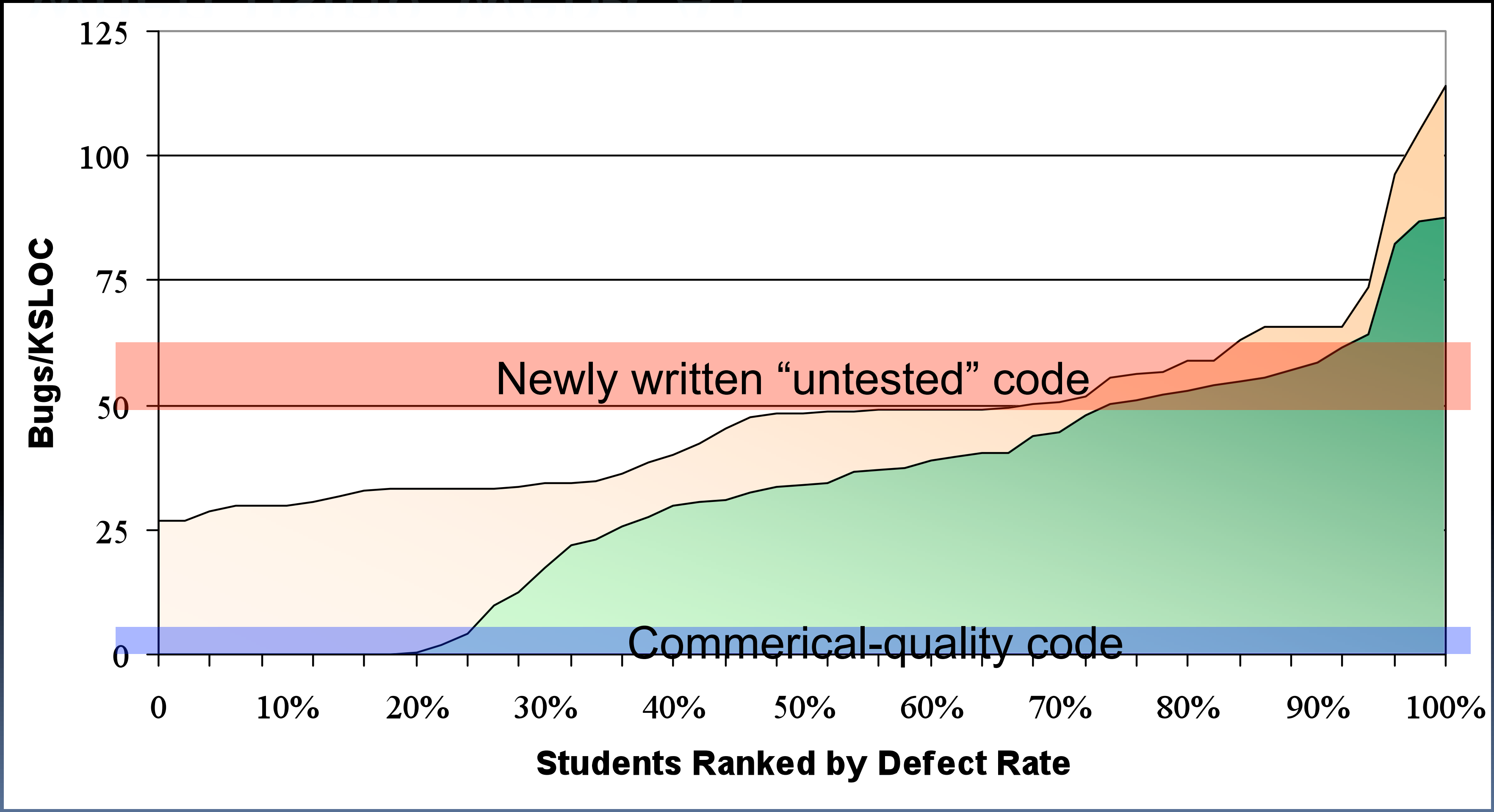
Error [Checkstyle]: -2
Missing a Javadoc comment.

```
118          {
119    7         turnLeft();
120    7         turnLeft();
121    7         turnLeft();
122          }
123
```

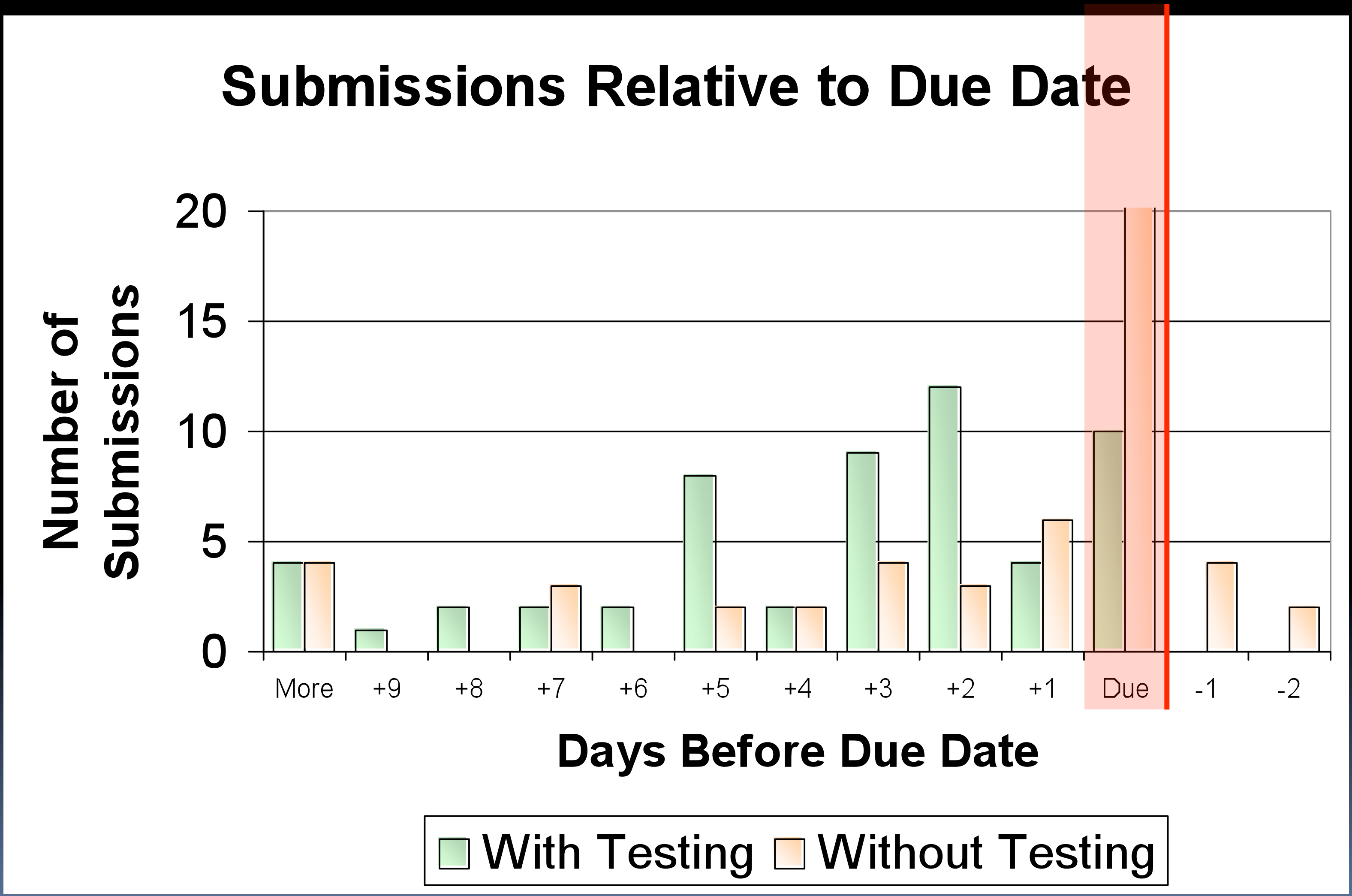
Assessing student tests is tricky, so we use complementary methods

- First, we measure how many of the student's own tests pass
- Second, we instrument student code and **measure code coverage** while the student's tests are running
- Third, we use instructor-provided **reference tests** to cross-check the student's tests
- We **multiply the percentages** together, so students must excel at all three to increase their score

Students improve their code quality when using Web-CAT



Students start earlier and finish earlier



Let's see it working!

- We'll walk through exactly how to get started
- Later, you can use the workshop materials from our SIGCSE 2009 workshop:

http://web-cat.org

Time for a break

- Any questions at this point?
- Anything in particular you definitely want to see me demonstrate?

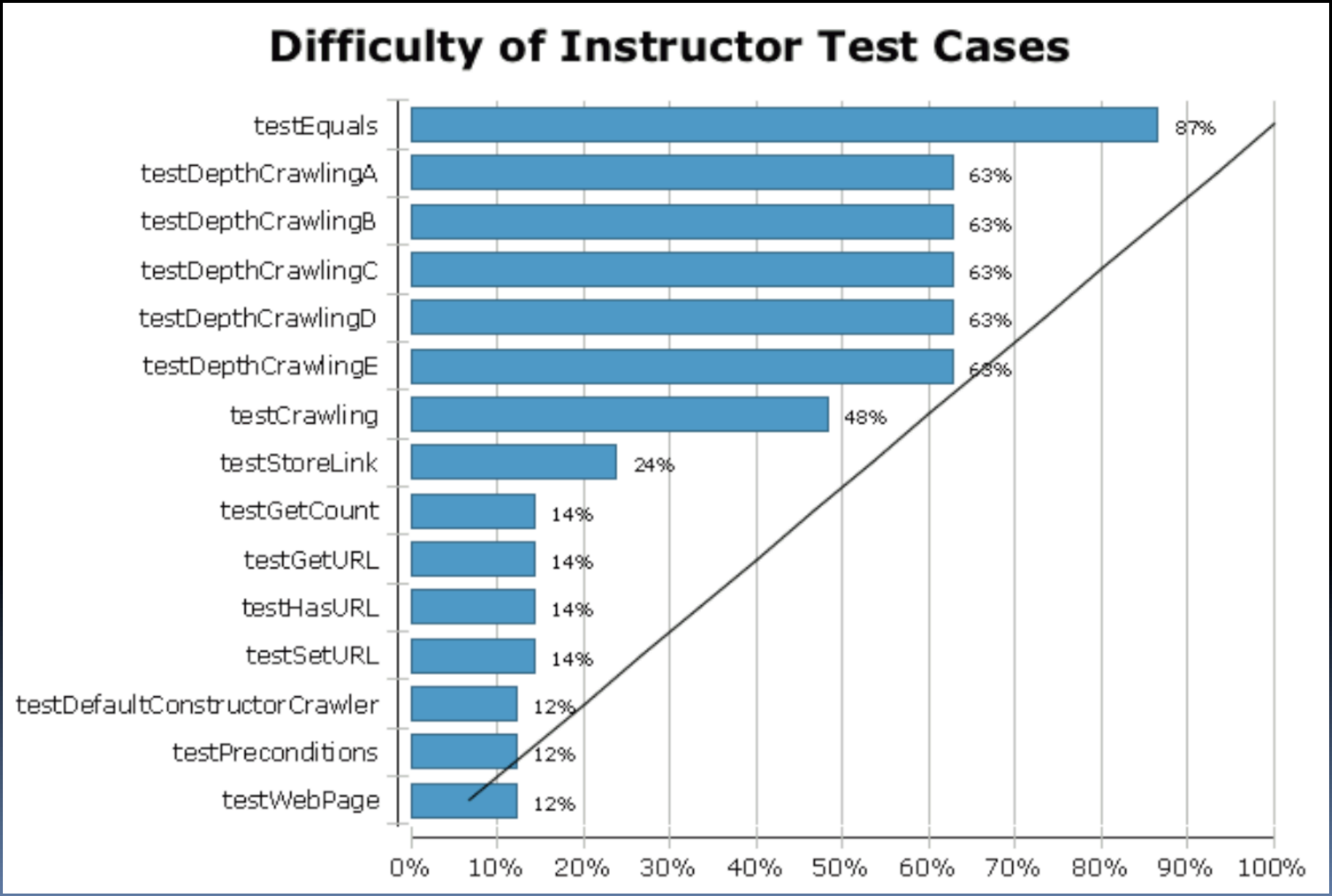
An inside peek at some “really cool things”!

- Testing stdout output
- What about stdin?
- Less strict comparisons
- Regular expressions, substrings
- Reflection-based tests

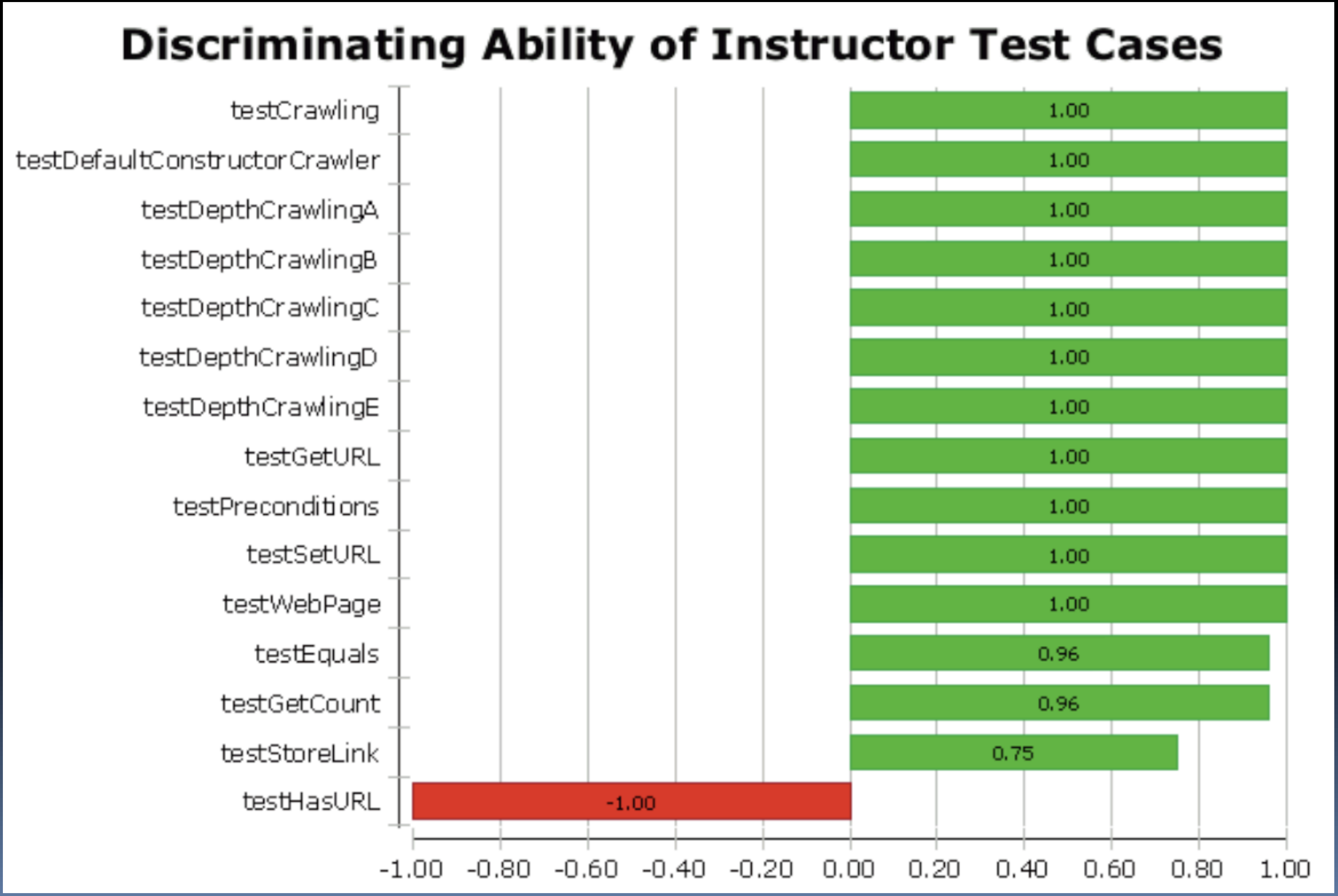
Walkthrough wrap-up

- Time for questions about the steps we have demonstrated …
- … or questions about how to use it with your own assignments

Assessing the difficulty of reference tests



Assessing reference test discrimination



The most important step in writing testable assignments is …

- Learning to write tests yourself
- Writing an instructor’s solution **with tests** that thoroughly cover all the expected behavior
- Practice what you are teaching/preaching
- Extra effort before assignment is “opened” (more prep time) but less effort after assignment is due (less grading time)

Lessons for writing assignments intended for automatic grading

- Requires greater clarity and specificity
- Requires you to explicitly decide what you wish to test, and what you wish to leave open to student interpretation
- Requires you to unambiguously specify the behaviors you intend to test
- Requires preparing a reference solution before the project is due, more upfront work for professors or TAs
- Grading is much easier as many things are taken care by Web-CAT; course staff can focus on assessing design

Areas to look out for in writing “testable” assignments

- How do you write tests for the following:
  - Main programs
  - Code that reads/write to/from stdin/stdout or files
  - Code with graphical output
  - Code with a graphical user interface

It is time for any final questions …

- About anything covered …
- About how we've used these techniques in courses
- About how we start our freshmen out in the very first lab
- About the availability of Web-CAT
- … Or anything else you want to ask

Visit our SourceForge project!

- http://web-cat.org/
- Info about using our automated grader, getting trial accounts, etc.
- Movies of making submissions, setting up assignments, and more
- Custom Eclipse and Visual Studio plug-ins for C++-style TDD
- Links to our own Eclipse feature site