Automatically Assessing Assignments That Use Test-Driven Development

Stephen H. Edwards
Virginia Tech, Dept. of CS
edwards@cs.vt.edu
http://people.cs.vt.edu/~edwards/

Overview

- **Need**:
  - Better software testing skills for our undergraduates
- **Idea**:
  - Require **test-driven development** in assignments across a variety of courses
- **Problem**:
  - How can we **assess** test suites to provide timely, effective feedback that encourages TDD?

Why a Comprehensive Approach?

- Students **cannot test their own code**
- Didn't want a single "testing course"
  - Probably optional to students
  - Upper division course has little impact on practices in other classes
  - Deeper training, but less impact on actions
- Instead, want a **culture shift** in what students actually **do**
- So, systematically incorporate testing expectations across many courses (including core)

Why TDD?

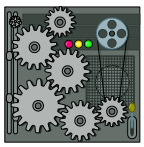
- Need something that:
  - Students can begin applying early
  - Students can directly relate to
  - Can grow with a student's abilities
- Provides immediate, almost visceral, benefits:
  - Increases confidence in correctness
  - Increases understanding of requirements
  - Jump starts incremental development
  - Preempts "big bang" integration problems

The Problem

- If we want students to do it, how do we:
  - **Assess** performance?
  - Give **productive feedback**?
  - Provide **rapid turnaround** needed?
- But, we don't want to:
  - Add to overloaded faculty
  - Require 2X (or more!) grading effort
  - Impose on underqualified TAs

Conventional Automated Assessment

- Many automated grading systems exist
- Basic strategy:
  - Student uploads program
  - Program is compiled
  - Executed against (instructor-provided) test data
  - Scored based on output
- Virginia Tech has one: the **Curator**
  - Servlet implementation
  - Allows multiple submissions
  - Allows random test data generation



Benefits of Conventional Approach

- Fast, precise **feedback** to students
- Chance(s) to **improve** based on feedback
- Students more likely to get a working solution
- More comprehensive assessment of correctness, so TAs can focus on design and style
- For us, instrumental in handling overpopulated core courses
- Did cause **culture change** without requiring extra class time or teaching effort
- **But …**

Limitations of Conventional Approach

- Students **focus on output correctness** as primary concern
- Result: students don't work on commenting, documentation (and even structure) until the end
- Students are **not encouraged or rewarded** for testing on their own
  - What's the benefit?
- Instead, students often **do less testing**
- Why test, if the auto-grader will do it for you?

Goals for Assessment

- Provide timely, useful feedback on **quality of tests** as well as quality of code
- Encourage students to write **thorough tests**
- Support rapid cycling:
  - "write a little test, write a little code"
- Employ a **grading/feedback/reward system** that fosters the behavior we want students to have

A New Strategy

- Require TDD test suite along with assignment
- Assess **test validity**: correctness of student's tests
- Assess **test completeness**: the "thoroughness" of student's tests
- Assess **program correctness**: behavior of student's solution

Choices

- Form of test cases
- Form of assignment submission
- Assessment approach
- Assessing test completeness
- Combining scores
- What information should be reported to students?
- Using independent test data

Piloting Pragmatics

- Want to pilot in a smaller class (that I'm teaching!)
- Available course: **Comparative Languages**
- Issues:
  - Programs in Pascal, Scheme, and Prolog
  - No conventional TDD tools
  - No direct analogs of OO TDD concepts
- Need a practical strategy to transition to core courses using Java or C++

Form of Test Cases

- Ideally, JUnit (or equivalent XUnit) tests
- Same form as the solution itself
- Can be uploaded along with solution

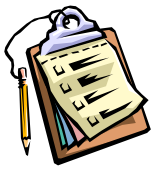
- For Languages course:
  - Use a plain ASCII file with simple markup to show start of each test case, input, and output

Form of Assignment Submission

- We are using a web application
- Ideal submission is a ZIP/JAR of the student's sources (program code and tests together)

- For Languages course:
  - All assignments involve a single source file
  - Students upload two separate files: "program" and "test data"

Assessment Strategy



- Use instructor-provided reference implementation
- Run student tests through reference imp. To assess **test validity**
- Also use the reference imp. to assess test completeness (more in a minute …)
- Finally, run student tests through student's program to assess **program correctness**

Assessing Test Completeness

- What is the best way to measure it?
- Instrument reference imp. to collect a coverage measure for **test completeness**
- Initial coverage measure: **branch coverage**
  - Seems to most closely match for TDD practices and recommendations
  - Easy for students to relate to
  - Still has black-box emphasis, since students do not have access to reference imp.

Combining Scores

- Three scores to combine:
  - Test validity
  - Test completeness
  - Program correctness
- Treat all three as percentages
- **Multiply them together**
- Requires an equal emphasis on all three
- Cannot neglect any one, since the neglected aspect drives the score arbitrarily low

Feedback to Students

- Besides the final score, what "results" does a student receive?
  - The three score components
  - Output similar to JUnit TextUI runner for student's program on student's tests
  - Same TextUI output for reference imp. on student's tests
  - Access to raw output files for both

Using Independent Test Data

- Should student program be tested against **instructor-provided** (or randomly generated) test data too?
- Our choice: **NO!**
  - Want to **empower** students with their own testing skills, **not invalidate** their tests by using "authoritative" alternatives
  - Want student to **own responsibility** for demonstrating correctness
  - Weighting formula includes coverage, so it is difficult to get a good score without good tests

Evaluation Plan

- Use it in Comparative Languages this semester
- Employ identical assignment used on "old" Curator (2 years ago)
- Compare total grades, "correctness" scores, non-submission rates
- Create a comprehensive test suite by hand and assess according to multiple coverage criteria
- Use this suite to assess latent bug densities in both collections and look for differences

Future Adaptation Plans

- Currently working on the necessary support to provide the same capability for Java + JUnit assignments
- Plans for C++ + CxxUnit support as well

- Current Curator design supports this expansion for multiple languages
  - Also easy to change coverage metric, scoring strategy, and other details as need arises

Summary

- Automatically assess TDD assignments using:
  - Reference implementation
  - Coverage instrumentation
  - Assess three dimensions:
    - Test validity
    - Test completeness
    - Program Correctness
- Key idea: **Feedback and scoring approach** that provides **positive incentive** for desired behavior

Questions?

Demo

- Let's try it!

Web - C A T help feedback log out

User: Stephen Edwards

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-Pascal
Submission no.:	9
File size:	14840
Submission time:	01/30/03 13:26
Deadline:	02/12/03 10:10
Late deadline:	02/14/03 10:10
Early bonus:	
Late penalty:	20.0 points per 1 days

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 v1.0: Testing your submission using p1tests.txt

.....F
case 6 FAILED: (no label)
......F
case 13 FAILED: (no label)
.................F
case 34 FAILED: (no label)
......

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

Test Validity (Reference Solution)

```
tddpas.pl v1.0: Testing reference implementation using p1tests.txt

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

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

- (•) Reference solution's output
- ( ) Your submission's output

View Selected File ...