A Leveled Examination of Test-Driven Development Acceptance

David Janzen
California Polytechnic State University
San Luis Obispo, CA

Hossein Saiedian
University of Kansas
Lawrence, KS
Outline

- Study Context
- Survey Questions
- Survey Results
- Confounding Factors
- Conclusions
- Future Work
Interest in Test-Driven Development

• TDD is a design (and testing) approach involving short, rapid iterations of
  Unit Test → Code → Refactor

• TDD is a key practice in XP
  – Emerging as a stand-alone practice for use with many processes

• May/June 2007 IEEE Software is a special issue on TDD
TDD Questions

• Should we adopt TDD?
  – Does TDD reduce defects? If so, at what cost?
  – Does TDD improve internal software quality?

• How do we teach TDD?

• When do we introduce TDD?
  – CS1, CS2, SE, MSE, Industry Training

• What do students think about TDD?
Original Study

• Series of leveled studies to compare test-first (TDD) and test-last approaches

• Hypothesis:
  – TDD improves internal software quality
    • Complexity, size, coupling, cohesion, testability

• Leveled:
  – CS1, CS2, Undergrad SE, Grad SE, Industry
This paper focuses on survey results.
<table>
<thead>
<tr>
<th>Course</th>
<th># Participants</th>
<th>Language</th>
<th>Collaboration</th>
<th>Project Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS1 Fall’05</td>
<td>80</td>
<td>C++</td>
<td>solo</td>
<td>2 weeks</td>
</tr>
<tr>
<td>CS2 Fall’05</td>
<td>36</td>
<td>C++</td>
<td>solo</td>
<td>2 weeks</td>
</tr>
<tr>
<td>CS2 Spring’06</td>
<td>38</td>
<td>C++</td>
<td>solo</td>
<td>2 weeks</td>
</tr>
<tr>
<td>Undergrad SE Summer’05</td>
<td>10</td>
<td>Java</td>
<td>teams of 3 or 4</td>
<td>16 weeks</td>
</tr>
<tr>
<td>Grad SE Fall’05</td>
<td>9</td>
<td>Java</td>
<td>teams of 3</td>
<td>16 weeks</td>
</tr>
<tr>
<td>Industry ’04–’06</td>
<td>12</td>
<td>Java</td>
<td>mostly solo, some pairing</td>
<td>3 hours in training, 4–6 months in study</td>
</tr>
</tbody>
</table>
Post-Study Survey Questions

• Choice: Which approach they would choose in the future?
• BestApproach: Which approach was the best for the project(s) they completed?
• ThoroughTesting: Which approach would cause them to more thoroughly test a program?
• Correct: Which approach produces a correct solution in less time?
• Simpler: Which approach produces code that is simpler, more reusable, and more maintainable?
• FewerDefects: Which approach produces code with fewer defects?
Survey Results

• Noted differences in survey results between early programmers (CS1/CS2) and more mature programmers (SE courses, Industry)

• What do you expect the differences to be?
  – Early programmers are open to TF/TL equally because they don’t have experience with either?
  – Mature programmers are more open to TL because that is what they are familiar with?
Reluctance to adopt test-first despite perceived benefits

11% vs 63% would choose test-first
Influence of TDD Experience

• Did using TDD influence programmer opinions regarding TDD perceptions?
3% vs 21% would choose test-first
40% vs 87% would choose test-first
Confounding Factors

• How much did the following affect programmer perceptions of TDD?
  – Programming language (C++/Java)
  – Automated testing framework (assert vs. JUnit)
  – Solo vs pair/team
  – Short (2 week) vs long (16 week) projects
## Late Breaking News

<table>
<thead>
<tr>
<th>Course</th>
<th># Students</th>
<th>Language</th>
<th>Collaboration</th>
<th>Project Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undergrad SE Capstone Fall’06–Spring’07</td>
<td>12</td>
<td>Java w/ GWT, JBoss, Spring, Hibernate</td>
<td>Teams of 5 or 6 with industry customer</td>
<td>30 weeks</td>
</tr>
</tbody>
</table>
Comments indicated challenges using test-first with new technologies

In line with 63% of other mature programmers
Conclusions

- Mature programmers may have higher opinions of TDD and be more likely to adopt TDD than early programmers
Speculation

• *Ubiquitous* TDD instruction may indoctrinate early programmers into thinking that TDD is the normal way to program
Test-Driven Learning

• Teach testing (and TDD) by example
  – Introduce and explore new concepts through automated unit tests in a test-first manner

• Inspired by
  – Testing Patterns In Beck’s “TDD: by Example”
    • Explanation Test
      – ask for and provide explanations in terms of tests
    • Learning Test
      – if you want to use a new method, class, or API, first write tests to learn how it works
## Test-Driven Learning in CS1/CS2

### Traditional Approach
```cpp
int sum(int min, int max) {
    int sum = 0;
    for(int i=min;i<=max;i++)
        sum += i;
    return sum;
}

int main() {
    cout << sum(3,7); //should print 25
    cout << sum(-2,2); //should print 0
    cout << sum(-4,-2); //should print -9
}
```

### TDL Approach
```cpp
int sum(int min, int max) {
    int sum = 0;
    for(int i=min;i<=max;i++)
        sum += i;
    return sum;
}

int main() {
    runTests();
}

void runTests() {
    assert(sum(3,7)==25);
    assert(sum(-2,2)==0);
    assert(sum(-4,-2)==-9);
}
```

---

Future Work

• Conduct same study in early programming course with Java and JUnit
• Apply TDL throughout early programming course
• Compare TDD with pairs and TDD with solo programmers
Questions?