Automated Unit Testing and the TSP

presented by Noopur Davis and Larry Maccherone

TSP Symposium
September 2007
Outline

- Background
- Changes to TSP
- Results from TSP teams
- Industry results
- Research topics
Unit Testing (UT)

IEEE definition of unit testing

*Testing of individual*
- *hardware or software units*
- *or groups of related units*

Common understanding of unit testing

- done by developers
- done on very small units of code
- goal is to ensure that isolated units of work are functioning correctly
Automated Unit Testing (AUT)

Automates the task of unit testing

- tests are usually written in the same language as production code.
- Tests are written to exercise units of code
  - in procedural languages, these could be functions and procedures
  - in object-oriented languages, these are frequently methods and classes.
Test Driven Development (TDD)

- What is TDD?
  - A strategy for software development where you write the tests before writing any production code
  - You expect the tests to fail the first time they are run
  - Tests serve as requirements or design artifacts
  - Force one to think about functionality and API before thinking about implementation

- TDD requires automated unit testing, but...
  - Not everyone doing automated unit testing is doing TDD
Why the Buzz?

- Most Agile methods strongly support automated unit tests, and some explicitly call out for TDD.
- Neither AUT nor TDD are new
  - The Agile community, in their own words, has “rediscovered” these
- “It is desirable to develop the tests before you write the code”. *A Discipline for Software Engineering*, page 370
Testing Frameworks

Automated unit tests are supported by testing frameworks that help with
- setup and teardown
- method and class-level testing
- family of assertions and generation of exceptions
- ability to extend the framework

The most popular family of frameworks is the x-Unit family (junit, nunit, cunit, phpunit, flexunit, etc..)
JUnit Example

// derived from example provided by Frank P. Coyle, PhD (http:// engr.smu.edu/~coyle)
public class LibraryTest extends TestCase {

    private Library library;

    public void setUp() throws Exception {
        library = new Library();
        library.addBook(new Book( "Cosmos", "Carl Sagan" ));
        library.addBook(new Book( "Contact", "Carl Sagan" ));
        library.addBook(new Book( "Contact", "Jena Malone" ));
    }

    public void tearDown() { library = null; }

    public void testGetBooksByTitle() {
        Vector books = library.getBooksByTitle( "Contact" );
        assertEquals( "wrong number of books found", 2, books.size() );
    }

    public void testGetBooksByAuthor() {
        Vector books = library.getBooksByAuthor( "Carl Sagan" );
        assertEquals( "2 books not found", 2, books.size() );
    }
    // Junit also provides assertTrue, assertFalse, assertNull, and a few more
}
Adding AUT to TSP

- Considerations for
  - Process framework
  - Planning framework
  - Quality framework
  - Measurement framework
Operational Details

Automated unit-tests are written along with production code in very tight increments

1. Write a couple of lines of production code*
2. Write a couple of lines of test code
3. Build and execute (most testing frameworks do this automatically)
4. Refactor both test and code if needed
5. Repeat

*for TDD, the order would be 2, 3, 1, 3, 4, 5
AUTs and Builds

AUTs must almost always be coupled with a build system that automatically builds and executes all unit tests (regression)

- Continuous builds
- Multiple builds a day
Process Considerations

"Out of the box" TSP

<table>
<thead>
<tr>
<th>Design</th>
<th>Design Review</th>
<th>Design Inspection</th>
<th>Code</th>
<th>Code Review</th>
<th>Code Inspection</th>
<th>Unit Test</th>
</tr>
</thead>
</table>

TSP with AUT

<table>
<thead>
<tr>
<th>Design</th>
<th>Design Review</th>
<th>Design Inspection</th>
<th>Production + Unit Test Code</th>
<th>Production + Unit Test Code Review</th>
<th>Production + Unit Test Code Inspection</th>
<th>Build/UT</th>
</tr>
</thead>
</table>

© Davis and Maccherone 12
## TSP Teams Data - LOC

<table>
<thead>
<tr>
<th>Name</th>
<th>Prod. LOC</th>
<th>UT LOC</th>
<th>UT Loc/Prod LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystem 1</td>
<td>1388</td>
<td>2126</td>
<td>1.53</td>
</tr>
<tr>
<td>Subsystem 2</td>
<td>1634</td>
<td>940</td>
<td>0.58</td>
</tr>
<tr>
<td>Subsystem 3</td>
<td>1863</td>
<td>1208</td>
<td>0.65</td>
</tr>
<tr>
<td>Subsystem 4</td>
<td>2009</td>
<td>1794</td>
<td>0.89</td>
</tr>
<tr>
<td>Subsystem 5</td>
<td>2667</td>
<td>781</td>
<td>0.29</td>
</tr>
<tr>
<td>Subsystem 6</td>
<td>3022</td>
<td>1442</td>
<td>0.48</td>
</tr>
<tr>
<td>Subsystem 7</td>
<td>3520</td>
<td>1851</td>
<td>0.53</td>
</tr>
<tr>
<td>Subsystem 8</td>
<td>4789</td>
<td>2197</td>
<td>0.46</td>
</tr>
<tr>
<td>Subsystem 9</td>
<td>7609</td>
<td>6125</td>
<td>0.80</td>
</tr>
<tr>
<td>Subsystem 10</td>
<td>12990</td>
<td>8481</td>
<td>0.65</td>
</tr>
<tr>
<td>Subsystem 11</td>
<td>17490</td>
<td>16233</td>
<td>0.93</td>
</tr>
<tr>
<td>Subsystem 12</td>
<td>55602</td>
<td>72269</td>
<td>1.30</td>
</tr>
</tbody>
</table>

**Average**: 0.76  
**Max**: 1.53  
**Min**: 0.29
# TSP Teams Data - Coverage

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Conditionals</th>
<th>Statements</th>
<th>Methods</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystem 1</td>
<td>97.6%</td>
<td>98.3%</td>
<td>100%</td>
<td>98.2%</td>
</tr>
<tr>
<td>Subsystem 2</td>
<td>50%</td>
<td>84.6%</td>
<td>95.3%</td>
<td>84.9%</td>
</tr>
<tr>
<td>Subsystem 3</td>
<td>66.9%</td>
<td>88.6%</td>
<td>91.5%</td>
<td>83.9%</td>
</tr>
<tr>
<td>Subsystem 4</td>
<td>62%</td>
<td>75.7%</td>
<td>89.6%</td>
<td>76%</td>
</tr>
<tr>
<td>Subsystem 5</td>
<td>40.7%</td>
<td>65.9%</td>
<td>80.5%</td>
<td>66.2%</td>
</tr>
<tr>
<td>Subsystem 6</td>
<td>66.6%</td>
<td>77%</td>
<td>83.5%</td>
<td>76.4%</td>
</tr>
<tr>
<td>Subsystem 7</td>
<td>60%</td>
<td>67.4%</td>
<td>63.9%</td>
<td>66%</td>
</tr>
<tr>
<td>Subsystem 8</td>
<td>66.7%</td>
<td>72.2%</td>
<td>100%</td>
<td>73.1%</td>
</tr>
<tr>
<td>Subsystem 9</td>
<td>76.7%</td>
<td>80.2%</td>
<td>100%</td>
<td>81.2%</td>
</tr>
</tbody>
</table>
## Industry Data – Microsoft TDD Case Study

<table>
<thead>
<tr>
<th></th>
<th>Windows</th>
<th>MSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test LOC/Source LOC</td>
<td>0.66</td>
<td>0.89</td>
</tr>
<tr>
<td>Block coverage</td>
<td>79%</td>
<td>88%</td>
</tr>
<tr>
<td>Development time (person months)</td>
<td>24</td>
<td>46</td>
</tr>
<tr>
<td>Team size</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>Decrease in Defects/LOC</td>
<td>38%</td>
<td>24%</td>
</tr>
<tr>
<td>Increase in dev time</td>
<td>25-35%</td>
<td>15%</td>
</tr>
</tbody>
</table>
## Industry Data – IBM Case Study

<table>
<thead>
<tr>
<th>Device driver</th>
<th>Test LOC/Prod LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device driver 1</td>
<td>.54</td>
</tr>
<tr>
<td>Device driver 2</td>
<td>.09</td>
</tr>
<tr>
<td>Device driver 3</td>
<td>.59</td>
</tr>
<tr>
<td>Device driver 4</td>
<td>.22</td>
</tr>
<tr>
<td>Device driver 5</td>
<td>.76</td>
</tr>
<tr>
<td>Device driver 6</td>
<td>.63</td>
</tr>
<tr>
<td>Device driver 7</td>
<td>.88</td>
</tr>
<tr>
<td>Device driver 8</td>
<td>1.12</td>
</tr>
<tr>
<td>Device driver 9</td>
<td>.13</td>
</tr>
<tr>
<td>Device driver 10</td>
<td>4.30</td>
</tr>
</tbody>
</table>
Results

- For TSP teams, the results as measured by improved system test defect density are inconclusive
  - The results are the “best in class” for TSP teams
  - They are not significantly better than other best in class teams that are not using AUT.

- The largest set of industry results from 19 case studies, controlled experiments, simulation, and artifact analysis shows*
  - Productivity decreased by 19% (-27% to 90%)
  - Quality improved by 25%

*Some data is based on “perception”
Lessons Learned

- Almost all serious testing efforts end up extending the test framework
- Not all tests can be automated
- Create a new test whenever a defect is detected that escaped the test suite
- Must have testable designs
- Hard to add to legacy
- AUTs result in “better” APIs, help document the code better, and do seem to help in code maintenance.
Planning Considerations

Size estimation
- Rule of thumb: plan to write as much unit test code as production code

Productivity
- Rule of thumb: unit test code is about 4 times faster to write than production code
- Plan for chunks of time to
  - Setup and learn test frameworks
  - Integrate AUTs into build system
  - Major re-factoring of tests every few iterations

Time-in-phase distribution
- Increase time in code phase
- Decrease time in unit test phase

Code coverage
- Most teams are aiming for 80%

*How can we get more empirical than “rules of thumb”? Larry will talk about this next.*
TSP Measurement Information Model

- Base measures:
  - Size, time, defects (, and schedule)
- Derived measures:
  - Simple ratios: defects/KLOC, LOC/hr, defect removal rates, etc.
  - Others: A/FR, Defect removal leverage, PQI, etc.
- Indicators/information product:
  - CR more efficient than UT at removing defects
  - Enough time spent on team inspection
  - Will (not) finish by planned completion date
Current TSP Information Model

- Historically effective
  - Encourages culture of review/inspection
  - Earned value tracking provides industry best progress feedback
- Adaptable to
  - Changes in process definition
  - Product as well as non-product activities
- Limitations with respect to emerging technology
  - Time spent writing automated unit tests should not be considered “failure”
  - Awkward or even misleading cost of quality formula
  - No ROI for future benefit
Unanswered Questions

- **Project questions:**
  - Cost/benefit of automated unit testing?
  - Effectiveness (or ineffectiveness) of automated code analysis?
  - How much automated unit testing and analysis to do?
  - Value of refactoring?
  - Reduce other appraisal activities in the presence of these? How much?

- **Process questions:**
  - New base measures needed? Or are simple bucketing and tagging changes sufficient?
  - What derived measures and indicators are necessary?
  - Do these proposed changes accommodate other new practices and technologies that are on the horizon?
Delayed Gratification

- Currently cost/benefit in TSP is only hinted at
  - Defects removed per time exerted in the current iteration
- A true ROI for defect prevention activities would compare two effort flows the same way we’d compare two cash flows in Economics 101.
Comparing Effort Flows

<table>
<thead>
<tr>
<th>Version 1</th>
<th>Version 2</th>
<th>...</th>
<th>Version N</th>
</tr>
</thead>
</table>

Vs.

<table>
<thead>
<tr>
<th>Version 1</th>
<th>Version 2</th>
<th>...</th>
<th>Version N</th>
</tr>
</thead>
</table>

- Design
- Reviews
- Prod. Code
- Test Code
- Unit test
- Other

© Davis and Maccherone
# Relationships

<table>
<thead>
<tr>
<th>Time invested in this</th>
<th>Should save time in this</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Code</td>
</tr>
<tr>
<td>Design</td>
<td>Fixing future defects</td>
</tr>
<tr>
<td>Review/inspection activities</td>
<td>Fixing future defects</td>
</tr>
<tr>
<td>Writing automated unit tests</td>
<td>Fixing future defects</td>
</tr>
<tr>
<td>Creating custom analysis rules</td>
<td>Fixing future defects</td>
</tr>
<tr>
<td>Writing automated unit tests and custom analysis rules</td>
<td>Refactoring</td>
</tr>
<tr>
<td>Refactoring</td>
<td>Adding new features</td>
</tr>
</tbody>
</table>

← The Agile message
How to Conduct Experiments?

1. Controlled experiment(s)
   - Have two groups (or more) develop the same thing in iterations. One with AUT, one without.

2. Longitudinially in a single project
   - Treat different parts of the code (but of same type) as separate efforts
   - Track effort in future iterations modifying or consuming those parts
   - This will require a very high level of traceability and automated data gathering from SCM, IDE, Build tools, etc.
Questions We Hope to Answer

- Production: Test code ratio:
  - What is the “ideal” ratio? 3:1? 1:1?
  - Does it matter what type of part you are building?
  - How do “time value of effort” calculations change the picture?
  - Is “ideal” different when Quality (as opposed to long-term cost/benefit) is of paramount concern?

- Coverage:
  - What is ideal coverage? 80%? Higher?
  - What characteristics indicate the need for higher or lower coverage?
Looking Ahead

- Automated gathering of data from IDEs, build tools, unit test output, analysis tool output
- Configurable tagging and historical relationship calculation
- Perspectives
  - Q: Do you count test writing as “Coding” or as “Testing” or what?
  - A: Count it as “Coding” if the current iteration is for writing tests. Count it as a Cost of Quality activity from the perspective of the entire project.
- Easy to use tools for teams to do flexible ex-post-facto analysis
References


Contact

Noopur Davis

- nd@sei.cmu.edu
- ndavis@davissys.com

Larry Maccherone

- LMaccherone@cmu.edu
- Larry@Maccherone.com