Leveraging Code Coverage Data to Improve Test Suite Efficiency and Effectiveness

Jean Hartmann, Ph.D.

Principal Test Architect
Engineering Systems Group
Microsoft Developer Division
Motivation
Goals
Benefits
Background
Stages of Analysis
Examples and Lessons Learned
Current Status of Pilot
Conclusion and Future Work
Why Optimize our Test Suites?

- Test suites are growing from release to release
- Increasing potential for test case redundancy
- Incur significant test execution, failure analysis and maintenance costs
- Number of product configurations to be validated is growing
- Excessive use of UI automation (unstable)

*Urgent need to ‘scrub’ our test suites!*
Goals

1. Improve test suite effectiveness
   - Identify unique contributions that specific tests or classes of tests make to exercising the product and raising their fault detection capability

2. Improve test suite efficiency
   - Identify a ‘core’ set of tests that maximizes product coverage
   - Support the analysis and identification of the potentially redundant or duplicate tests
Benefits

- Increased test agility
  - Focus test resources on a significantly smaller set of tests that maximizes product coverage
  - Ensure that a larger number of product configurations can be tested more thoroughly
  - Reduce costs for execution, analysis and maintenance
  - Increase time on task for QA (doing test design!)
- Increased tribal knowledge and understanding of test case purpose, content, priority and product coverage
- Leverage a solution (process and supporting tools) that can be applied on a per release or more frequent basis
Background: DevDiv Testing

- DevDiv produces Visual Studio, Visual Studio Team System and Expression Suite
- Current (VS2010) ship cycle includes 10,000s of automated tests (legacy and new)
- Mostly C++/C# test code for API/UI automation
- Mix of Dev unit and QA tests
- Most stored in Maddog Test Case Management (TCM) system
Background: Devdiv Code Coverage

- Based on proprietary Magellan tool set
- Binary instrumentation to measure block/arc coverage
- Coverage DB is created to store results and enable results merging across different product builds
- Results typically collected on per test basis (‘trace’)
- Maddog test metadata is associated with each trace, e.g. test id, test priority, execution time, etc.
- Tool set performance/scalability enables vast amounts of coverage data to be collected
- Used during ship cycles to report compliance with MS code coverage exit criteria (70% block coverage)
Stage 1: Test Suite Characterization

- Code coverage % – not focusing on test suite adequacy, but product coverage
- Explore unique and cumulative % coverage contributions by test priority
- Identify ‘sweet spot’ where cumulative coverage contributions across product flatten out, while number of tests increases (coverage delta)
- Gauge ratio of core set vs. potential dupe set. The higher the ratio, the more effective the current core set at exercising the product, e.g. for P0 tests, ratio = 319/453 (~0.7)
Based on sweet spot, examine unique coverage contributions of P2 and P3 tests (using prev. example)

Extract P2/P3 tests that uniquely exercise specific functions/methods and code blocks exercised

Rank tests according to their unique contributions

Actions for team: set threshold, review and discuss and reprioritize valuable tests (e.g. P2/3->P0/1). Use ranking data in conjunction with team’s domain expertise
Stage 3: Determine Core Set

- Leverages a Magellan tool known as Scout
- **Scout inputs:**
  - Test Traces (from coverage DB)
- **Scout outputs XML reports:**
  - Summary identifying core set of tests across product binaries
  - Details per product binary
- Reports can easily be transformed into Maddog (TCM) queries for execution
- Filtering of traces prior to Scout analysis is possible using regex on metadata
- Processing typically takes .5 to 1 hour for larger test suites
Scout: Maxcoverage Algorithm

Denotes that a trace T covers the impacted block

Number is a maximum overall coverage for that trace

Result mask: 6 6 6 5 6 6 5 6

Final Set: T1 T4 T6
Stage 4: Review Potential Duplicates

- Core set of tests is not a min set
- Remaining (non-core) tests constitute potential set of duplicate tests (dupes)
- Core tests and dupes are assigned to each feature team owner for review/analysis
- Feature team owners pay special attention to:
  - Regression bug finders
  - Extensive verifications – keep or merge
  - Sub-scenarios – keep or merge
  - Boundary tests
  - Special cases (e.g. data-driven tests containing SQL statements)
  - Scenario order differences – execution order not preserved by trace
- Review core test trace using Sleuth (coverage viewer) to examine and compare trace differences
- Review and compare core and dupe test code (often using simple regex tools)
- Keep notes in spreadsheets for peer review
- Team decides on a dupe elimination strategy, e.g. disable in Maddog until next major milestone, then delete?
One product team within VS with approx. 20 binaries

Statistics for recent VS2010 test pass:

- 12500 tests yielding 22000 results (98% UI tests)
- Execution time for full run: 5 days on 60 machines
- Failure analysis effort: 2 weeks for a team of 19 (FTE + vendors)
- Bugs found: 33 product bugs + 38 (UI auto) test bugs

Code coverage data was collected on a per result basis using Magellans tool suite and kept in DB

Trace metadata annotated with test priority and pass/fail data in preparation for test suite characterization and to enable Scout pre-analysis filtering
### Results for a Specific Feature Team

<table>
<thead>
<tr>
<th>CATEGORY</th>
<th>TOTAL</th>
<th>% of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total # of tests (P0/1/2/3) for this team</td>
<td>1629</td>
<td>100%</td>
</tr>
<tr>
<td>Core test set identified by Maxcoverage</td>
<td>746</td>
<td>46%</td>
</tr>
<tr>
<td>Potential dupe set identified (remainder)</td>
<td>883</td>
<td>54%</td>
</tr>
<tr>
<td>Updated dupe set (after review)</td>
<td>715</td>
<td>44%</td>
</tr>
<tr>
<td>Final set (MaxCoverage+(potential-actual))</td>
<td>914</td>
<td>56%</td>
</tr>
</tbody>
</table>

Final set lies within 10% of computed Maxcoverage set
Reasons for Duplicates

- Timeframe - 10 years of adding tests
- Test planning process
  - Creating individual scenario tests, but never reviewing and merging them into larger scenario tests
- Test code review process
  - Incomplete review process that has led to junk being checked-in, e.g. Test case skeleton created with setup/teardown, but empty test body!
TC1: Verify table in database shows up in Server Explorer
TC2: Verify View Designer (if open) after choosing New View
TC3: Verify you can drag a table from Server Explorer to View Designer
TC4: Verify you can check table columns on the View Designer
TC5: Verify you can execute query from the View Designer selecting table data

TC6-10: Above 5 scenarios repeated, this time for the QUERY Designer

=> Review led to merging above tests into two longer scenario tests
Lessons Learned by Teams

- Get cleaner in stages...
  - Milestone exits
  - Feature RI quality gates
  - Test pass preparation phase
  - Milestone Quality (MQ) – start of new dev. cycle
- ..and then stay clean!
  - More careful test plan designs and reviews
    - Examine new and existing tests and take appropriate action, e.g. merge tests where appropriate
    - Consider regression possibilities
    - Add new tests judiciously using coverage data as a feedback loop
  - Improve test code review processes
  - Become real experts of owned areas and tests that exercise them (domain experts)
Approx. 20 teams enrolled in pilot

Most pilot teams are reviewing and analyzing their core/dupe sets

Preliminary results indicate significant potential for savings (red)

Actual savings shown in green

Some factors influencing actual savings:
- Availability of domain experts
- Overlap with regression bug finding tests
- Time since last clean-up effort
- Process maturity and adherence

Exploring advanced tools (SPIRIT) that also considers temporal and static analysis data to reduce dupe analysis time

Working on additional tools/utilities to help compare and contrast remaining dupes

<table>
<thead>
<tr>
<th>VS Team</th>
<th>Total No. of Tests</th>
<th>Core Test Set Identified by Tool</th>
<th>Final Set (after Review)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team 1</td>
<td>11913</td>
<td>2304 (20%)</td>
<td>In Progress</td>
</tr>
<tr>
<td>Team 2</td>
<td>12040</td>
<td>1095(9%)</td>
<td>In Progress</td>
</tr>
<tr>
<td>Team 3</td>
<td>436</td>
<td>209(48%)</td>
<td>390(90%)</td>
</tr>
<tr>
<td>Team 4</td>
<td>650</td>
<td>180(28%)</td>
<td>200(31%)</td>
</tr>
<tr>
<td>Team 5</td>
<td>2719</td>
<td>504(18%)</td>
<td>In Progress</td>
</tr>
<tr>
<td>Team 6</td>
<td>4400</td>
<td>1226(27%)</td>
<td>In Progress</td>
</tr>
<tr>
<td>Team 7</td>
<td>4027</td>
<td>1386(34%)</td>
<td>In Progress</td>
</tr>
<tr>
<td>Team 8</td>
<td>580</td>
<td>365(63%)</td>
<td>In Progress</td>
</tr>
<tr>
<td>Team 9</td>
<td>1293</td>
<td>232(14%)</td>
<td>In Progress</td>
</tr>
</tbody>
</table>
Need to preserve our test agility and reduce testing resources now!

Test suite optimization needs to become integral part of our lifecycle activities

Pilot will continue and analyze ROI (incl. from additional techniques such as weighted Maxcoverage and greedy algorithms)

Vision – collection and analysis is mostly automated with minor input from testers
Resources

- SPIRIT paper: http://portal.acm.org/citation.cfm?doid=1595696.1595748
- PNSQC09 paper supporting this talk
- Related papers on Scout usage