Early and Cost-Effective Software Fault Detection

Lars-Ola Damm
Lars-ola.damm@ericsson.com

Overview (1/2)
- Research setting
  - Conducts research as an industrial PhD student
  - 50/50 employment at BTH and Ericsson AB
  - 80% research, 20% project work at Ericsson AB
  - Research conducted since Jan. 2003

Overview (2/2)
- Background (initial research question)
  - "Why is the test lead-time so long and how can it be shortened?"

Research focus (1/2)
- Licentiate thesis title:
  - Monitoring and Implementing Early and Cost-Effective Software Fault Detection
  - Faults should be found earlier to the extent it is cost-effective (not all faults are the cheapest to find early in relation to the required test effort)
  - The right faults should be found in the right phase

Research focus (2/2)
- Study existing research methods
- Determine which are practically useful
- Identify refinements that are needed
- Determine for which purposes each method is feasible

Method/Research process
- The Licentiate thesis comprised a set of case studies that addressed two main areas:
  - Test automation framework
  - Methods for fault-based process improvement
- Enhanced fault analysis method
- Implementation of test framework
- Redesign of framework implementation
- Enhanced for monitoring support
Results so far (overview)

1. Method for identifying improvement opportunities
2. Framework for finding more faults earlier
3. Enhanced method for process assessment and improvement
4. Approach for monitoring improvement implementations

Results so far (1/4)

• Method for identifying improvement opportunities
• Fault-slip-through definition, measure
• Cost of fault-slip-through, improvement potential
• Baseline measure – identification of key area to improve

Phase Found:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Unit Test</th>
<th>Function Test</th>
<th>System Test</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belonging/phase</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit Test</td>
<td>4</td>
<td>25</td>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>Function Test</td>
<td></td>
<td>15</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>System Test</td>
<td></td>
<td></td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>Operation</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Tot. found/phase</td>
<td>4</td>
<td>40</td>
<td>36</td>
<td>8</td>
</tr>
</tbody>
</table>

Fault-slip-through

All faults found during operation are fault-slip-through
Net fault-slip-through (belong to same phase as when found)
Not relevant

% fault slippage measures can then be obtained from the previous table:

<table>
<thead>
<tr>
<th>% Faults-slip-through</th>
<th>Project X</th>
</tr>
</thead>
<tbody>
<tr>
<td>To Function Test</td>
<td>25/40=62%</td>
</tr>
<tr>
<td>To System Test</td>
<td>(18+5)/36=64%</td>
</tr>
</tbody>
</table>

Average cost of faults in different phases (example):

Avoidable cost: 50-25=25
Avoidable cost: 10-2=8
Results so far (1/4)

- Method for identifying improvement opportunities
- Faults-slip-through definition, measure
- Cost of faults-slip-through (improvement potential)
- Baseline measure of identification of key area to improve

Percent improvement potential by phase:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Percent</th>
<th>Baseline</th>
<th>Improvement</th>
<th>Potential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial release</td>
<td>56.5%</td>
<td>56.5%</td>
<td>0%</td>
<td>56.5%</td>
</tr>
<tr>
<td>Final release</td>
<td>62.5%</td>
<td>62.5%</td>
<td>0%</td>
<td>62.5%</td>
</tr>
<tr>
<td>Total potential</td>
<td>60.5%</td>
<td>60.5%</td>
<td>0%</td>
<td>60.5%</td>
</tr>
</tbody>
</table>

Results so far (2/4)

- Framework for finding more faults earlier
- Component level test automation
- Component level test-driven development
- Results from implementation in two development projects

Percent faults-slip-through:

<table>
<thead>
<tr>
<th>Product A</th>
<th>Percent</th>
<th>Component A</th>
<th>Component B</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT</td>
<td>69%</td>
<td>69%</td>
<td>69%</td>
</tr>
<tr>
<td>ST</td>
<td>49%</td>
<td>49%</td>
<td>49%</td>
</tr>
<tr>
<td>FSF</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
</tr>
</tbody>
</table>

Results so far (3/4)

- Framework for finding more faults earlier
- Component level test automation
- Component level test-driven development
- Results from implementation in two development projects

Percent faults-slip-through:

<table>
<thead>
<tr>
<th>Product B</th>
<th>Percent</th>
<th>Component A</th>
<th>Component B</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT</td>
<td>76%</td>
<td>76%</td>
<td>76%</td>
</tr>
<tr>
<td>ST</td>
<td>55%</td>
<td>55%</td>
<td>55%</td>
</tr>
<tr>
<td>FSF</td>
<td>56%</td>
<td>56%</td>
<td>56%</td>
</tr>
</tbody>
</table>

Results so far (4/4)

- Framework for finding more faults earlier
- Component level test automation
- Component level test-driven development
- Results from implementation in two development projects

Percent faults-slip-through:

<table>
<thead>
<tr>
<th>Product C</th>
<th>Percent</th>
<th>Component A</th>
<th>Component B</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>ST</td>
<td>60%</td>
<td>60%</td>
<td>60%</td>
</tr>
<tr>
<td>FSF</td>
<td>61%</td>
<td>61%</td>
<td>61%</td>
</tr>
</tbody>
</table>

Note: The fault-slipage costs decreased by about 2-3 times in comparison to the previous project.
## Results so far (3/4)

- Enhanced method for process assessment and improvement
  - Faults-slip-through combined with fault trigger classification
- Definition: A fault trigger indicates which type of test activity that triggered the fault, i.e., the type of test activity that was necessary to expose the fault.

**Example:**

<table>
<thead>
<tr>
<th>Trigger type</th>
<th>Description</th>
<th>Nr faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal Interface</td>
<td>Faults in interfaces between components</td>
<td>10</td>
</tr>
<tr>
<td>External Interface</td>
<td>Faults in external interfaces, e.g., protocols</td>
<td>5</td>
</tr>
<tr>
<td>Human Interface</td>
<td>Faults in user interfaces, e.g., GUI</td>
<td>3</td>
</tr>
<tr>
<td>Performance</td>
<td>Performance faults</td>
<td>2</td>
</tr>
<tr>
<td>Faults</td>
<td>Stability faults, includes load and endurance problems</td>
<td>5</td>
</tr>
<tr>
<td>Configuration</td>
<td>Software and hardware configuration faults, e.g., installation faults, includes version faults</td>
<td>7</td>
</tr>
</tbody>
</table>

The purpose is to determine which activities in which phases need to be improved.

## Results so far (4/4)

- Approach for monitoring improvement implementation
  - Faults-slip-through with continuous goal follow-up
- Objectives with investigating monitoring support
  - Obtain early feedback for decision support
  - Increase the amount of manual analysis work
  - Increase the visibility of the measurement results

## Limitations of the results

- Benefits of test frameworks such as the one presented here are dependent on the architecture of the product
- Test-driven development does however not seem to have any specific limitations
- The fault analysis methods are most likely not useful in:
  - Organizations that do not have a well defined test process and do not have a formalized and tool supported fault handling process
  - Products with very few faults (e.g., small projects)
Further work

- Replicated studies on the applicability of the techniques, e.g. to determine which contexts and purposes they are suitable for.
- Investigate module-directed fault classification, e.g. which modules are more fault prone than others?
- Further investigate implications of applying techniques in products developed in geographically dispersed locations.
- Enhanced support for improvement monitoring.
- Develop a framework for early and cost-effective fault detection.

Further information

Lars-Ola Damm
lars-ola.damm@ericsson.com

Dante could never have painted a more fearsome hell than the last few months of a large software project which did not make extensive use of unit testing. There are few creatures more feral and dangerous than a stressed, sleep-deprived, overly-caffeinated programmer who has just been told that a bug they fixed two months ago has resurfaced... for the third time. I've been there. More than once. I'm not going back.

[Krishna Kumar 2001]