Practical Issues in Measuring Software Quality

Allen P. Nikora
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109-8099
Allen.P.Nikora@jpl.nasa.gov
Motivation

- Over the past several years, techniques have been developed to:
  - Estimate a software component’s proportional fault burden
  - Use measures of a software component’s structural evolution to estimate fault insertion rates
  - Estimate test effectiveness
  - Use risk factors to estimate impacts of a change to a system’s quality

- Practical issues:
  - Measuring software structural change
  - Fault identification
  - Obtaining profile information
Measuring Structural Evolution

Program → Metric Analysis → CMA → Raw Metrics → Principal Components Analysis → Relative Complexity

- Modules

- Metric Analysis

- Raw Metrics:
  - 12 23 54 12 203 39 238 34
  - 7 13 64 12 215 9 39 238
  - :
  - 11 21 54 12 241 39 238 35
  - 5 33 44 12 205 9 138 44
  - 42 55 54 12 113 29 234 14

- Principal Components Analysis

- Relative Complexity:
  - 50
  - 40
  - :
  - 60
  - 45
  - 55
Relative Complexity

- Relative complexity is a synthesized metric
  \[ \rho_i^B = \sum_{j=1}^{m} \lambda_j^B d_j^B \]

- Relative complexity is a fault surrogate
  - Composed of metrics closely related to faults
  - Highly correlated with faults
Measuring Software Evolution
Comparing Two Builds

Source Code → Baseline → Baseline Build i → RCM Values → RCM Delta → Code Churn

Source Code → Baseline → Baseline Build j → RCM Values → RCM Delta → Code Deltas
Measuring Evolution

- Different modules in different builds
  - $M^{i,j}_a$ set of modules not in latest build
  - $M^{i,j}_b$ set of modules not in early build
  - $M^{i,j}_c$ set of common modules
- Code delta $\delta_a^{i,j} = \rho^{B,j}_a - \rho^{B,i}_a$
- Code churn $\chi_a^{i,j} = |\delta_a^{i,j}| = |\rho^{B,j}_a - \rho^{B,i}_a|$
- Net code churn $\nabla^{i,j} = \sum_{m_c \in M_c} \chi_c^{i,j} + \sum_{m_a \in M^{i,j}_a} \rho^{B,i}_a + \sum_{m_b \in M^{i,j}_b} \rho^{B,j}_b$
Estimating Fault Insertion Rate

- Proportionality constant, $k'$, representing the rate of fault insertion
- For $j^{th}$ build, total faults inserted

$$F^j = kR^0 + k'\Delta^{0,j}$$

- Estimate for the fault insertion rate

$$F^{j+1} - F^j = kR^0 + k'\nabla^{0,j+1} - kR^0 + k'\nabla^{0,j}$$

$$= k'(\nabla^{0,j+1} - \nabla^{0,j})$$

$$= k'\nabla^{j,j+1}$$
Identifying and Counting Faults

- Unlike failures, faults are not directly observable
- Fault counts should be at the same level of granularity as software structure metrics
- Failure counts could be used as a surrogate for fault counts if:
  - Number of faults were related to number of failures
  - Distribution of number of faults per failure had low variance
  - The faults associated with a failure were confined to a single procedure/function

Actual situation shown on next slide
Observed Distribution of Faults per Failure

Distribution of Faults per Failure

<table>
<thead>
<tr>
<th>Frequency</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faults per Reported Failure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Statistics

<table>
<thead>
<tr>
<th>Defects per 1 Failure</th>
<th>N Valid</th>
<th>Missing</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Deviation</th>
<th>Percentiles 25</th>
<th>Percentiles 50</th>
<th>Percentiles 75</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
<td>0</td>
<td>10.5667</td>
<td>7.5000</td>
<td>9.3428</td>
<td>3.7500</td>
<td>7.5000</td>
<td>13.2500</td>
</tr>
</tbody>
</table>
Fault Identification and Counting

- Faults must be identified at the module level.
- To calibrate the regression model for fault insertion rates, for each fault repaired:
  - Determine the point at which it was first inserted into the module (e.g., inserted for version i of module A).
  - Compute the structural change between versions i and i-1 of module A.
Fault Identification and Counting

- Rules have been developed to identify and count faults in source code.
- Tracing faults to their points of insertion becomes easier if there are links between the CM system and the problem reporting system (i.e., for a specific problem report, what source files were changed, and which versions of each source file repaired the fault?)
- Changes due to enhancements must be separated from changes due to fault repair
Estimating Test Efficiency

- Measures of structural evolution can be used together with profile information to estimate test efficiency.
  - Ideal profile - computed from cumulative structural change of modules since last test
  - Actual profile - observed during test execution
- Issue - instrumenting embedded real-time software system to obtain execution profile during test
Obtaining Execution Profile

- Build instrumentation into system
- Compile instrumentation into system
- Build execution profile logging capability into multi-mission simulator
  - For bit-level simulators being considered, appears to be specific instance of breakpoint capability
  - Behavior of software under test will not change (timing relationships will not be affected by instrumentation compiled into system)
  - Becomes part of institutional infrastructure, rather than being a project-to-project effort.