Developing Fault Models for Space Mission Software

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

John C. Munson
Computer Science Department
University of Idaho
Moscow, ID
jmunson@cs.uidaho.edu

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology. This work is sponsored by the National Aeronautics and Space Administration's Office of Safety and Mission Assurance under the NASA Software Program led by the NASA Software IV&V Facility. This activity is managed locally at JPL through the Assurance Technology Program Office (ATPO).
Outline

- Goal
- Measurement Requirements
  - Measuring Structural Evolution
  - Identifying and Counting Faults
- Modeling Fault Content
- Current and Future Work
- Discussion and Conclusions
- References and Further Reading
Goal: Improve Understanding of Fault Generation Process

Look for relationships between:
- Measurements of a system’s structure.
- Number of faults inserted during development.

Recent work
- Classification Methods
  - Classification Trees [Ghok97]
  - Regression Trees [Khos01a]
- Discriminant Functions
  - Boolean Discriminant Functions [Schn97]
  - Relative Critical Value Deviation [Schn01a]
- Regression Modeling
  - Zero-inflated Poisson Regression [Khos01]
  - Logistic Regression [Schn01]

Limitation – these efforts look at the system at a particular moment in time
May limit validity of results at other points during development effort
Goal: Improve Understanding of Fault Generation Process (cont’d)

Analyzing SW Evolution

- Examines entire developmental history of a SW system
- Conclusions valid over entire development interval analyzed
- Measurement requirements
  - Quantifiable notion of software evolution
  - Quantitative definition of software fault
Required Measurement Characteristics

Measurement mechanisms must have the following characteristics:

- Satisfy representation condition [Fen97]
- Produce accurate, repeatable, consistent measurements
- Measurements of artifacts, fault counts, development process made at same level of detail
- Perceived benefit of making measurements must exceed cost of making them
Measuring Structural Evolution

Measure change activity between successive versions of the system.

Granularity Issues

- Structural evolution is measured at the module level
- Every version of every module is measured.
- Changes between subsequent versions of a module are measured with respect to a chosen baseline.

Distinct sources of variation identified with Principal Components Analysis (PCA) [Dill84]

Measurements are performed automatically outside of the development environment

- Minimize impact of measurement activities on developers.
- Relies on read access to CM repository.
- Accomplished with Darwin network appliance [Cyla03].
## Module Attributes

<table>
<thead>
<tr>
<th>Metric</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec</td>
<td>Number of executable statements</td>
</tr>
<tr>
<td>NonExec</td>
<td>Number of non-executable statements</td>
</tr>
<tr>
<td>$N_1$</td>
<td>Total operator count</td>
</tr>
<tr>
<td>$\eta_1$</td>
<td>Unique operator count</td>
</tr>
<tr>
<td>$N_2$</td>
<td>Total operand count</td>
</tr>
<tr>
<td>$\eta_2$</td>
<td>Unique operand count</td>
</tr>
<tr>
<td>Nodes</td>
<td>Number of nodes in the module control flow graph</td>
</tr>
<tr>
<td>Edges</td>
<td>Number of edges in the module control flow graph</td>
</tr>
<tr>
<td>Paths</td>
<td>Number of paths in the module control flow graph</td>
</tr>
<tr>
<td>MaxPath</td>
<td>The length of the path with the maximum edges</td>
</tr>
<tr>
<td>AvePath</td>
<td>The average length of the paths in the module control flow graph</td>
</tr>
<tr>
<td>Cycles</td>
<td>Total number of cycles in the module control flow graph</td>
</tr>
</tbody>
</table>

Standardized definitions were developed for each measurement – see [Cyla03]
# Principal Components of Raw Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Exec</td>
<td>.60</td>
</tr>
<tr>
<td>NonExec</td>
<td>.64</td>
</tr>
<tr>
<td>$N_1$</td>
<td>.28</td>
</tr>
<tr>
<td>$\eta_1$</td>
<td>.49</td>
</tr>
<tr>
<td>$N_2$</td>
<td>.28</td>
</tr>
<tr>
<td>$\eta_2$</td>
<td>.35</td>
</tr>
<tr>
<td>Nodes</td>
<td>.87</td>
</tr>
<tr>
<td>Edges</td>
<td>.88</td>
</tr>
<tr>
<td>Paths</td>
<td>.17</td>
</tr>
<tr>
<td>MaxPath</td>
<td>.87</td>
</tr>
<tr>
<td>AvePath</td>
<td>.86</td>
</tr>
<tr>
<td>Cycles</td>
<td>.67</td>
</tr>
</tbody>
</table>

| Eigenvalues | 4.79  | 3.13  | 2.24  |

Measurement domains resulting from PCA
The Measurement Process

Build i
Source Code

Measurement Tools

Baseline

Baselined Build i

PCA Domain Scores

Domain Score Change

Baselined Build j

Domain Churn

Domain Deltas
View of Structural Evolution at the System Level

Graph of Code Churn and Code Delta for the project fdms_project_1b.
View of Structural Evolution at the Module Level

(Non-zero) Modules for build 2001-11-02 of project project_1b, sorted by Churn since baseline.

<table>
<thead>
<tr>
<th>ModuleName</th>
<th>Churn From Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>doContext(XML_Parser, int startTagLevel, const ENCODING *enc, const char *s, const char *end, const char **nextPtr)</td>
<td>329.523165</td>
</tr>
<tr>
<td>doProlog(XML_Parser, const ENCODING *enc, const char *s, const char *end, int tok, const char *next, const char **nextPtr)</td>
<td>311.585820</td>
</tr>
<tr>
<td>examples()</td>
<td>309.099387</td>
</tr>
<tr>
<td>processMeasurementAndPredict(const Mds::Fw:Time:Tsng:RTEpoch&amp; current, const Mds::Fw:Time:Tsng:RTEpoch&amp; stop)</td>
<td>289.353008</td>
</tr>
<tr>
<td>test2s()</td>
<td>279.141671</td>
</tr>
<tr>
<td>TestDiscrete::TestDiscrete()</td>
<td>260.394345</td>
</tr>
<tr>
<td>TestIntervale::TestIntervale()</td>
<td>256.865234</td>
</tr>
<tr>
<td>storeAttr(XML_Parser, const ENCODING *enc, const char *attStr, TAG_NAME *tagNamePtr, BNDING **bindingsPtr)</td>
<td>240.951958</td>
</tr>
<tr>
<td>GreaseFilterTest(Dispatch&amp;r, const std::string&amp; key, const CGIArgs&amp; args)</td>
<td>240.699311</td>
</tr>
<tr>
<td>doTest()</td>
<td>237.759572</td>
</tr>
<tr>
<td>PositionEstimateFunctionTest(Dispatch&amp;r, const std::string&amp; key, const CGIArgs&amp; args)</td>
<td>223.900440</td>
</tr>
<tr>
<td>DirectedGraph::close()</td>
<td>214.416659</td>
</tr>
<tr>
<td>SimpleNormalPositionEstimatorTraits::Thread::updateStateVariables(const Mds::Fw:Time:Tsng:RTEpoch&amp; start, const Mds::Fw:Time:Tsng:RTEpoch&amp; stop, const Mds::Fw:Filter::Grease::GreaseBasis&amp; state, const Mds::Fw:Filter::Grease::GreaseBasis&amp; covariance, const Mds::Red::Mars::Common::SimpleAuDragModel::AuDragModelParameterType&amp; au_drag_model_para, double spacecraft_mass, double avg_engine_thrust)</td>
<td>204.313726</td>
</tr>
<tr>
<td>AuDragModelParameterEstimatorTraits::Thread::predictState()</td>
<td>203.781925</td>
</tr>
<tr>
<td>ParachuteEstimatorTraits::Thread::predictState()</td>
<td>195.118089</td>
</tr>
<tr>
<td>SimpleNormalPositionEstimatorTraits::Thread::changed(const Mds::Fw::Cmp::RefCountComponentInstance monitored_nv, Mds::Fw:DM:Vhis::ConstItemVectorRef changed_items)</td>
<td>182.312589</td>
</tr>
<tr>
<td>PREFIX(prologTok)</td>
<td>175.659645</td>
</tr>
<tr>
<td>LengthTest(Dispatch&amp;r, const std::string&amp; key, const CGIArgs&amp; args)</td>
<td>165.230353</td>
</tr>
<tr>
<td>vmain(int cior, const char* argList)</td>
<td>164.506422</td>
</tr>
<tr>
<td>GoalNetTestHarness: createXGoalNet(Dispatch&amp;r, const std::string&amp; key, const CGIArgs&amp; args)</td>
<td>163.479793</td>
</tr>
<tr>
<td>vmain_internal(const char* argList)</td>
<td>162.467209</td>
</tr>
</tbody>
</table>
Identifying and Counting Faults

Accurate software fault prediction depends on precise, measurable definition of a fault. Until 2002, no definition of fault in measurable terms:

- IEEE Standards
- ODC [Chil92]
- Previous work (Annual Oregon Workshop on Software Metrics, May 11-13, 1997) [Niko97]
Fault Enumeration

- Examine changes made in response to reported failures
- Base recognition/enumeration of software faults on the grammar of the software system's language
- Fault measurement granularity in terms of tokens that have changed [Muns02]
Fault Enumeration (cont'd)

Consider each line of text in each version of the program as a bag of tokens

- If a change spans multiple lines of code, all lines for the change are included in the same bag

Number of faults based on bag differences between

- Version of program exhibiting failures
- Version of program modified in response to failures

Use version control system to distinguish between

- Changes due to repair and
- Changes due to functionality enhancements and other non-repair changes
Fault Enumeration: Example

Example 1

- Original statement: $a = b + c$;
  
  $B_1 = \{<a>, \leq, <b>, \leq, <c>\}$

- Modified statement: $a = b - c$;
  
  $B_2 = \{<a>, \leq, <b>, \geq, <c>\}$

- $B_1 - B_2 = \{\leq, \geq\}$

- $|B_1| = |B_2|$, $|B_1 - B_2| = 2$

- One token has changed $\Rightarrow 1$ fault
Fault Enumeration: Example

Example 2

- Original statement: \( a = b - c; \)
  \[ B_3 = \{<a>, <=>, <c>, <->, <b>\} \]
- Modified statement: \( a = 1 + c - b; \)
  \[ B_4 = \{<a>, <=>, <1>, <+>, <c>, <->, <b>\} \]
- \( B_3 - B_4 = \{<1>, <+>\} \)
- \( |B_3| = 6, |B_4| = 8, |B_4| - |B_3| = 2 \)
- 2 new tokens representing 2 faults
Modeling Fault Content

Fault models developed from:

- Measured structural evolution (cumulative amount of change for each module).
- Number of faults repaired for each module.

Analysis indicates that the amount of structural evolution is related to the number of faults repaired [Niko03].
Modeling Results

Regression ANOVA

<table>
<thead>
<tr>
<th>Source</th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>10091546</td>
<td>3</td>
<td>3363848</td>
<td>293</td>
<td>P&lt;.01</td>
</tr>
<tr>
<td>Residual</td>
<td>6430656</td>
<td>560</td>
<td>11483</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>16522203</td>
<td>563</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Regression Model

<table>
<thead>
<tr>
<th>Model</th>
<th>Coefficients</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Constant)</td>
<td>18.24</td>
<td>3.5</td>
<td>P&lt;.01</td>
</tr>
<tr>
<td>Domain 1 Churn</td>
<td>21.63</td>
<td>17.3</td>
<td>P&lt;.01</td>
</tr>
<tr>
<td>Domain 2 Churn</td>
<td>-.59</td>
<td>-.3</td>
<td>p&gt;.01</td>
</tr>
<tr>
<td>Domain 3 Churn</td>
<td>.93</td>
<td>.7</td>
<td>p&gt;.01</td>
</tr>
</tbody>
</table>

Quality of the Regression Model

<table>
<thead>
<tr>
<th>Model Summary</th>
<th>R</th>
<th>R Square</th>
<th>Adjusted R Square</th>
<th>Std. Error of the Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.782</td>
<td>.611</td>
<td>.609</td>
<td>107.16024</td>
</tr>
</tbody>
</table>
Current and Future Work

- Determine whether the fault insertion rate changes over time
- Identify which types of faults correspond to which types of change.
- Expand the number of projects to obtain larger measurement baseline.
  - Collaborating with GSFC SATC to infuse measurement techniques
- Enlarge the set of structural measurements taken by network appliance (e.g., include CDK OO measures [Chid94]).
- Include effect of measurable development process characteristics in fault models
- Integrate measurement of system execution profile to estimate
  - Test effectiveness
  - Risk of exposure to residual faults
Current and Future Work (cont’d)

- Estimate amount of noise in fault counts
  - Not all changes associated with a PR may actually be repairs
  - “Pocket PRs”
    - Not a significant issue for this development effort because of how CM is set up and discipline of development team
    - May be issue for other efforts
  - Unequal test coverage – some components may be more heavily tested, finding more faults
Current and Future Work (cont’d)

- Resolve known fault counting inaccuracies
  - Example 1 – adding operators/operands
    - Original faulty statement: $a = b + c$;
    - Repaired statement: $a = b - c + d$;
    - Bag difference: $\{\langle-\rangle, <d>\}$
    - 3 tokens added or changed, however
  - Example 2 – token reordering
    - Original faulty statement: $a = b - c$;
    - Repaired statement: $a = c - b$;
    - Bag difference: $\{\}$
    - Number of reordered tokens cannot be accurately determined
- Develop training materials for practitioners
  - Measurement user’s guide
  - Measurement tutorials
Discussion and Conclusions

At least for the data with which we have worked, a software component's fault burden is related to the measured amount of change during its development. Practical techniques for measuring structural evolution and the number of repaired faults have been developed and are available for use on "real" development efforts.

- Allows development of fault models that can provide additional information to help decide where to deploy fault identification and repair resources (e.g., test staff, additional inspections).
References and Further Reading


References and Further Reading (cont’d)


References and Further Reading (cont’d)


