Infusing Software Fault Measurement and Modeling Techniques

Allen Nikora
Software Quality Assurance Group
Jet Propulsion Laboratory
California Institute of Technology

Al Gallo
Software Assurance Technology Center
Goddard Space Flight Center
Greenbelt, MD

John Munson
Department of Computer Science
University of Idaho
Moscow, ID

NASA Code Q Software Program Center Initiative UPN 323-08; Kenneth McGill, Research Lead

OSMA Software Assurance Symposium 2003
July 30 – Aug 1, 2003

The work described in this presentation 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).
Overview
Goals
Benefits
Approach
Status
Current Results
Papers and Presentations Resulting From This CI
Overview

Objectives: Gain a better quantitative understanding of the effects of requirements changes on fault content of implemented system. Gain a better understanding of the type of faults that are inserted into a software system during its lifetime.

Use measurements to PREDICT faults, and so achieve better

- planning (e.g., time to allocate for testing, identify fault prone modules)
- guidance (e.g., choose design that will lead to fewer faults)
- assessment (e.g., know when close to being done testing)

<table>
<thead>
<tr>
<th>Structural Measurements of Specification</th>
<th>Structural Measurements of Source Code</th>
<th>Estimated Fault Counts by Type for Implemented System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Specifications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Source Code Modules</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lines of Source Code</td>
<td>Max Nesting Depth</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>375</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>258</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>Source Code Modules</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Operands</td>
<td>Total Nesting Depth</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

- Conditional Execution Faults
- Execution Order Faults
- Incorrect Computation
- Variable Usage Faults

Numbers of estimated faults of given type in given module

<table>
<thead>
<tr>
<th>Estimated Fault Counts by Type for Implemented System</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 1 1.5</td>
</tr>
<tr>
<td>3 1.5 1</td>
</tr>
<tr>
<td>3.5 1.25 1.9</td>
</tr>
</tbody>
</table>

types of measurements
- Function Count
- Number of Environmental Constraints
- exceptions
- types of measurements
- Environmental Constraints
Goals

Develop a viable method of infusing the measurement and fault modeling techniques developed during the first two years of this task into software development environments at GSFC and JPL

- Collaborate with SATC and selected projects at GSFC
- Continue and extend collaboration with projects at JPL

Develop training materials for software measurement for software engineers/software assurance personnel

- Measurement background
- Using DARWIN Network Appliance
- Organizational interfaces
- Interpreting output
Benefits

- Provide quantitative information as a basis for making decisions about software quality.
- Use easily obtained metrics to identify software components that pose a risk to software and system quality.
- Measurement framework can be used to continue learning as products and processes evolve.
Approach

Measure structural evolution on collaborating development efforts
  ♦ Structural measurements for several JPL projects collected
  ♦ Several GSFC projects have shown interest

Analyze failure data
  ♦ Identify faults associated with reported failures
    ♦ Relies on:
      • All failures being recorded
      • Failure reports specifying which versions of which files implement changes responding to the reported failure.

  ♦ Count number of repaired faults according to token-count technique reported in ISSRE’02 [Mun02]. (Fault count is dependent variable)

Analyze relationships between number of faults repaired and measured structural evolution during development
Identify relationships between requirements change requests and implemented quality/reliability
- Measure structural characteristics of requirements change requests (CRs).
- Track CR through implementation and test
- Analyze failure reports to identify faults inserted while implementing a CR
Develop training materials for software measurement for software engineers/software assurance personnel
- DARWIN user’s guide nearly complete
- Measurement class materials being prepared
Follow-on to previous 2-year effort, “Estimating and Controlling Software Fault Content More Effectively”.

- Investigated relationships between requirements risk and reliability.
- Installed improved version of structural and fault measurement framework on JPL development efforts
  - Participating efforts
    - Mission Data System (MDS)
    - Mars Exploration Rover (MER)
    - Multimission Image Processing Laboratory (MIPL)
    - GSFC efforts
  - All aspects of measurement are now automated
    - Fault identification and measurement was previously a strictly manual activity
  - Measurement is implemented in DARWIN, a network appliance
    - Minimally intrusive
    - Consistent measurement policies across multiple projects
Current Results: Measuring Software Structural Evolution

- Mars Exploration Rover (MER)
- Multimission Image Processing Laboratory (MIPL)
- Mission Data System (MDS)
  - Structural measurements collected for release 5 of MDS
    - > 1500 builds
    - > 65,000 unique modules
- Domain scores, "domain churn", and proportional fault burdens computed
  - At system level
  - At individual module level
- > 1,400 anomaly reports analyzed
Current Results: Measuring Software Structural Evolution

Build i

Source Code

Measurement Tools

Baseline

PCA Domain Scores

Domain Score Change

Baselined Build i

Domain Churn

Build j

Baselined Build j

Domain Deltas
The goal of the Darwin web portal is to provide a solid, easy to use interface to the Darwin system. Contained in this web portal you can find Manager Information, Tester Information, Darwin Education, and Project Management.

This is the main page of the DARWIN measurement system’s user interface.
Chart of a system's structural evolution during development. This is available under "Manager Information". Clicking on a data point will bring up a report detailing the amount of change that occurred in each module. This plot shows some of the individual builds for release 5 of the MDS.
This report shows the amount of change that's occurred for each module shown in this particular build (2002-02-07).
## Current Results: Measuring Software Structural Evolution

<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.
## Current Results: Measuring Software Structural Evolution

<table>
<thead>
<tr>
<th>Metric</th>
<th>Domain 1</th>
<th>Domain 2</th>
<th>Domain 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec</td>
<td>0.60</td>
<td>0.49</td>
<td>0.47</td>
</tr>
<tr>
<td>NonExec</td>
<td>0.64</td>
<td>0.53</td>
<td>0.18</td>
</tr>
<tr>
<td>$N_1$</td>
<td>0.28</td>
<td>0.64</td>
<td>0.65</td>
</tr>
<tr>
<td>$\eta_1$</td>
<td>0.49</td>
<td>0.70</td>
<td>0.07</td>
</tr>
<tr>
<td>$N_2$</td>
<td>0.28</td>
<td>0.64</td>
<td>0.65</td>
</tr>
<tr>
<td>$\eta_2$</td>
<td>0.35</td>
<td>0.90</td>
<td>0.04</td>
</tr>
<tr>
<td>Nodes</td>
<td>0.87</td>
<td>0.31</td>
<td>0.27</td>
</tr>
<tr>
<td>Edges</td>
<td>0.88</td>
<td>0.31</td>
<td>0.27</td>
</tr>
<tr>
<td>Paths</td>
<td>0.17</td>
<td>-0.10</td>
<td>0.89</td>
</tr>
<tr>
<td>MaxPath</td>
<td>0.87</td>
<td>0.35</td>
<td>0.29</td>
</tr>
<tr>
<td>AvePath</td>
<td>0.86</td>
<td>0.34</td>
<td>0.33</td>
</tr>
<tr>
<td>Cycles</td>
<td>0.67</td>
<td>0.22</td>
<td>-0.02</td>
</tr>
<tr>
<td>Eigenvalues</td>
<td>4.79</td>
<td>3.13</td>
<td>2.24</td>
</tr>
</tbody>
</table>

Table above shows measurement domains resulting from PCA
Current Results: Fault Identification and Measurement

Developing software fault models depends on definition of what constitutes a fault

- Desired characteristics of measurements, measurement process
  - Repeatable, accurate count of faults
  - Measure at same level at which structural measurements are taken
    - Measure at module level (e.g., function, method)
  - Easily automated

More detail in [Mun02]
Current Results: Fault Identification and Measurement

Approach

- 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
Current Results: Fault Identification and Measurement

Approach (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
Current Results: Fault Identification and Measurement

Example 1

- Original statement: \( a = b + c; \)
  \( B_1 = \{<a>, <\rightarrow>, <b>, <\rightarrow>, <c>\} \)
- Modified statement: \( a = b - c; \)
  \( B_2 = \{<a>, <\rightarrow>, <b>, <\leftarrow>, <c>\} \)
- \( B_1 - B_2 = \{<\rightarrow>, <\leftarrow> \} \)
- \( |B_1| = |B_2|, |B_1 - B_2| = 2 \)
- One token has changed \( \Rightarrow \) 1 fault
Current Results: Fault Identification and Measurement

Example 2

- Original statement: \( a = b - c; \)
  \( B_2 = \{\langle a\rangle, \langle >=\rangle, \langle b\rangle, \langle <=\rangle, \langle c\rangle\} \)
- Modified statement: \( a = c - b; \)
  \( B_3 = \{\langle a\rangle, \langle >=\rangle, \langle c\rangle, \langle <=\rangle, \langle b\rangle\} \)
- \( B_2 - B_3 = \{ \} \)
- \( |B_2| = |B_3| , |B_2 - B_3| = 0 \)
- 1 fault representing incorrect sequencing
Current Results: Fault Identification and Measurement

Example 3

- Original statement: \( a = b - c \);
  \[ B_3 = \{<a>, \langle>, <c>, \langle>, <b>\} \]
- Modified statement: \( a = 1 + c - b \);
  \[ B_4 = \{<a>, \langle>, <1>, \langle>, <c>, \langle>, <b>\} \]
- \( B_3 - B_4 = \{<1>, \langle>\} \)
- \(|B_3| = 6\), \(|B_4| = 8\), \(|B_4| - |B_3| = 2\)
- 2 new tokens representing 2 faults
Current Results: *Fault Identification and Measurement*

Available Failure/Fault Information

- For each failure observed during MDS testing, the following information is available
  - The names of the source file(s) involved in repairs
  - The version number(s) of the source files in repairs
- Example on next slide
# Current Results: Fault Identification and Measurement

## Available Failure/Fault Information – Example

<table>
<thead>
<tr>
<th>Directory</th>
<th>File name</th>
<th>Version</th>
<th>Problem Report ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>CurrentTime.cpp</td>
<td>1</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>make.cfg</td>
<td>4</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>make.cfg</td>
<td>3</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>make.cfg</td>
<td>2</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>RTDuration.cpp</td>
<td>2</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>RTDuration.h</td>
<td>2</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>RTEpoch.cpp</td>
<td>2</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>RTEpoch.h</td>
<td>2</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>TestRTDuration.cpp</td>
<td>0</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>TestRTDuration.cpp</td>
<td>1</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>TestRTDuration.cpp</td>
<td>0</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>TestRTDuration.h</td>
<td>2</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>TestRTDuration.h</td>
<td>1</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>TestRTDuration.h</td>
<td>0</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>testRTEpoch.cpp</td>
<td>1</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>TmgtException.cpp</td>
<td>0</td>
<td>IAR-00182</td>
</tr>
<tr>
<td>MDS_Rep/source/Mds/Fw/Time/Tmgt/c++</td>
<td>TmgtException.h</td>
<td>0</td>
<td>IAR-00182</td>
</tr>
</tbody>
</table>
Current Results: Fault Identification and Measurement

Fault Identification and Counting Tool Output

MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ArchetypeConnectorFactory.cpp 1 42
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ArchitectureInstanceRegistry.cpp 1 79
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ArchitectureInstanceRegistry.cpp 2 8
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ArchitectureInstanceRegistry.cpp 3 0
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ArchManagedInstance.cpp 1 36
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ CallableInterface.cpp 1 49
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ CGIMethodRegistration.cpp 1 4
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ComponentComponentLinkInstance.cpp 1 0
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ComponentComponentLinkInstance.cpp 2 65
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ComponentConnectorLinkInstance.cpp 1 0
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ComponentConnectorLinkInstance.cpp 2 50
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ComponentObjectLinkInstance.cpp 1 27
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ComponentObjectLinkInstance Arguments.cpp 1 0
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ComponentRegistration.cpp 1 2
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ConcreteComponentInstance.cpp 1 8
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ConcreteComponentInstance.cpp 2 0
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ConcreteConnectorInstance.cpp 1 42
MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++ ConcreteConnectorInstance.cpp 2 27

Output format:

<Source file name> <source file version> <fault count>
Current Results: 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 [Nik03].
Current Results: Modeling Fault Content

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 Results: Modeling Fault Content

Fault Counting Method vs. Model Quality

Which fault counting methods produce better fault models?

- Number of tokens changed
- Number of “sed” commands required to make each change
- Number of modules changed
Current Results: *Modeling Fault Content*

Fault Counting Method vs. Model Quality

### Number of tokens changed

<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;.05</td>
</tr>
<tr>
<td>Domain 1 Churn</td>
<td>21.63</td>
<td>17.3</td>
<td>p&lt;.05</td>
</tr>
<tr>
<td>Domain 2 Churn</td>
<td>-0.59</td>
<td>-0.3</td>
<td>p&gt;.05</td>
</tr>
<tr>
<td>Domain 3 Churn</td>
<td>0.93</td>
<td>0.7</td>
<td>p&gt;.05</td>
</tr>
</tbody>
</table>

### Number of “sed” commands

<table>
<thead>
<tr>
<th>Model</th>
<th>Coefficients</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Constant)</td>
<td>2.484</td>
<td>14.555</td>
<td>p&lt;.05</td>
</tr>
<tr>
<td>Domain 1 Churn</td>
<td>0.151</td>
<td>3.411</td>
<td>p&lt;.05</td>
</tr>
<tr>
<td>Domain 2 Churn</td>
<td>0.529</td>
<td>6.489</td>
<td>p&lt;.05</td>
</tr>
<tr>
<td>Domain 3 Churn</td>
<td>-0.087</td>
<td>-1.791</td>
<td>p&gt;.05</td>
</tr>
</tbody>
</table>

### Number of modules changed

<table>
<thead>
<tr>
<th>Model</th>
<th>Coefficients</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Constant)</td>
<td>1.200</td>
<td>35.995</td>
<td>p&lt;.05</td>
</tr>
<tr>
<td>Domain 1 Churn</td>
<td>0.009</td>
<td>1.041</td>
<td>p&gt;.05</td>
</tr>
<tr>
<td>Domain 2 Churn</td>
<td>0.143</td>
<td>8.920</td>
<td>p&lt;.05</td>
</tr>
<tr>
<td>Domain 3 Churn</td>
<td>-0.043</td>
<td>-4.483</td>
<td>p&lt;.05</td>
</tr>
</tbody>
</table>
Current Results: Modeling Fault Content

Fault Counting Method vs. Model Quality

<table>
<thead>
<tr>
<th>Model</th>
<th>R Square</th>
<th>Adjusted R Square</th>
<th>Std. Error of the Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1 – tokens changed</td>
<td>0.61</td>
<td>0.061</td>
<td>107.16</td>
</tr>
<tr>
<td>Model 2 – number of &quot;sed&quot; commands</td>
<td>0.19</td>
<td>0.19</td>
<td>3.88</td>
</tr>
<tr>
<td>Model 3 – number of changed modules</td>
<td>0.14</td>
<td>0.14</td>
<td>0.76</td>
</tr>
</tbody>
</table>

Comparison of Model Quality
Current Results: User’s Guide

Table of Contents

1. Introduction
   1.1. Purpose
   1.2. Scope

2. Measuring a Software System’s Structural Evolution
   2.1. Data Requirements
   2.2. Creating a New Project in Darwin
   2.3. Importing the Source Repository
   2.4. Initial Structural Measurement
   2.5. Analyzing Structural Evolution
      2.5.1. Establishing a Baseline
      2.5.2. Computing Structural Change
   2.6. Structural Measurement Outputs

3. Measuring the Number of Faults Repaired
   3.1. Data Requirements
   3.2. Identifying Changes
   3.3. Counting Faults
   3.4. Fault Measurement Outputs
   3.5. Potential Sources of Noise

4. Developing Fault Models
   4.1. Data Requirements
   4.2. Module-Level Fault Models
   4.3. System-Level Fault Models

5. References

APPENDICES

A. Importing and Measuring a Source Repository
   A.1. Creating a Development Environment Copy of the Repository
   A.2. Creating a Copy of the Repository on Darwin
      A.2.1. Creating the Repository Structure
      A.2.2. Copying Files Into the Repository
   A.3. Measuring a System’s Evolution

B. Identifying and Counting Repaired Faults
   B.1. LogSpecifiedVersionDiffs.pl
   B.2. strip.pl
   B.3. bagdiff4.pl
   B.4. final.pl
   B.5. extract.pl
   B.6. CPPLexer.pm
Papers and Presentations Resulting From This CI


Papers and Presentations Resulting From This CI


