Estimating and Controlling Software Fault Content More Effectively

Allen P. Nikora
Autonomy and Control Section
Jet Propulsion Laboratory
California Institute of Technology

Norman F. Schneidewind
Department of Information Sciences
Naval Postgraduate School
Monterey, CA

John C. 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
Sept 4-6, 2002

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).
Agenda

- Overview
- Goals
- Benefits
- Approach
- Status
- Current Results
- References
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)

Structural Measurements of Specification

Structural Measurements of Source Code

Estimated Fault Counts by Type for Implemented System

SAS'02
Goals

- Quantify the effects of requirements changes on the fault content of the implemented system by identifying relationships between measurable characteristics of requirements change requests and the number and type of faults inserted into the system in response to those requests.
- Improve understanding of the type of faults that are inserted into a software system during its lifetime by identifying relationships between types of structural change and the number and types of faults inserted.
- Improve ability to discriminate between fault-prone modules and those that are not prone to faults.
Benefits

- Use easily obtained metrics to identify software components that pose a risk to software and system quality.
  - Implementation – identify modules that should have additional review prior to integration with rest of system
  - Prior to implementation – estimate impact of changes to requirements on quality of implemented system.
- Provide quantitative information as a basis for making decisions about software quality.
- Measurement framework can be used to continue learning as products and processes evolve.
Approach

- Measure structural evolution on collaborating development efforts
  - Initial set of structural evolution measurements collected
- Analyze failure data
  - Identify faults associated with reported failures
  - Classify identified faults according to classification rules
  - Identify module version at which each identified fault was inserted
  - Associate type of structural change with fault type
Approach (cont’d)

- 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
**Approach: Structural Measurement Framework**

1. **Problem Reports**
   - Extract Repaired Source Files
   - Identify Source Files Repaired
   - Repaired File IDs
   - Compare Repairs to Faulty Files
   - Fault Regions

2. **Fault Identification and Handling Rules**
   - Identify Faults
   - Discovered Faults
   - Find Initial Fault Occurrence
   - Initial Fault Placement
   - Add fault placement to repository

3. **Baseline Measurements**
   - OM Library
   - Extract changed source files
   - Most recently changed source files
   - Measure most recently changed source files
   - Raw structural measurements
   - Add structural measurements to repository

4. **Measurement Baseline**
   - Compute fault index
   - Fault indices
   - Place fault indices into repository
   - Measurement Repository
   - module name, revision number, fault index

5. **Fault Burden**
   - Compute Proportional Fault Burden
   - Proportional Fault Burden
   - module name, revision number, fault index
   - Develop fault content regression model
   - Regression coefficients
   - Compute absolute fault burden
   - Absolute Fault Burden

**SAS’02**
Year 2 of planned 2-year study
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)

- All aspects of measurement framework shown on slide 8 can now be 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: Requirements Risk vs. Reliability

- Analyzed attributes of requirements that could cause software to be unreliable
  - Space
  - Issues
- Identified thresholds of risk factors for predicting when number of failures would become excessive
- Further details in [Schn02]
Current Results:
Requirements Risk vs. Reliability

Cumulative Failures vs. Cumulative Memory Space

CF = 6E-07*CS^2 - 0.0003*CS + 1.9511

CF: Cumulative Failures
CS: Cumulative Memory Space
Current Results: Requirements Risk vs. Reliability

CF = 0.2481860 * (exp(0.0107263 * CI))

Cumulative Failures vs. Cumulative Issues

CF: Cumulative Failures
CI: Cumulative Issues
Current Results: Requirements Risk vs. Reliability

Rate of Change of Failures with Memory Space

\[
CF = (-0.0003) + (0.0000012 \times CS)
\]

CF: Cumulative Failures
CS: Cumulative Memory Space
Current Results: Requirements Risk vs. Reliability

Rate of Change of Failures with Issues

dCF/dCI = 0.0107263 * CF

CF: Cumulative Failures
CI: Cumulative Issues

SAS'02
Current Results:
Requirements Risk vs. Reliability

\[ CS = \frac{b+(b^2-(4*a*(c-CF)))^{0.5}}{2*a} \]

- Predicting cumulative risk factors
  - Cumulative memory space vs. cumulative failures

SAS’02
Current Results:
Requirements Risk vs. Reliability

Cumulative Issues

Cumulative Failures

- Predicting cumulative risk factors
  - Cumulative issues vs. cumulative failures

CI = LN(CF/a)/b
a = 0.248186
b = 0.0107263
CF: Cumulative Failures
CI: Cumulative Issues

SAS'02
Current Results: *Fault Types vs. Structural Change*

- Structural measurements collected for release 5 of Mission Data System (MDS)
  - 1,828 source files
  - 66,063 unique modules
  - 2,197,916 total measurements made
- Fault index and proportional fault burdens computed
  - At system level
  - At individual module level
- Next slides show typical outputs of DARWIN network appliance
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 (2001-03-10).
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: $a = b + c$;
    - $B_1 = \{<a>, <=>, <b>, <+>, <c>\}$
  - Modified statement: $a = b - c$;
    - $B_2 = \{<a>, <=>, <b>, <->, <c>\}$
  - $B_1 - B_2 = \{<+>, <->\}$
  - $|B_1| = |B_2|$, $|B_1 - B_2| = 2$
  - One token has changed $\Rightarrow$ 1 fault
Current Results: *Fault Identification and Measurement*

- **Example 2**
  - Original statement: \( a = b - c \);
    - \( B_2 = \{<a>, <=>, <b>, <->, <c>\} \)
  - Modified statement: \( a = c - b \);
    - \( B_3 = \{<a>, <=>, <c>, <<->, <b>\} \)
  - \( B_2 - B_3 = \{\} \)
  - \(|B_2| = |B_3|, |B_2 - B_3| = 0\)
  - 1 fault representing incorrect sequencing
Current Results: *Fault Identification and Measurement*

- **Example 3**
  - Original statement: \( a = b - c; \)
    - \( B_3 = \{<a>, \langle=\rangle, <c>, \langle-\rangle, <b>\} \)
  - Modified statement: \( a = 1 + c - b; \)
    - \( B_4 = \{<a>, \langle=\rangle, <1>, \langle+\rangle, <c>, \langle-\rangle, <b>\} \)
  - \( B_3 - B_4 = \{<1>, \langle+\rangle\} \)
  - \( |B_3| = 6, |B_4| = 8, |B_4| - |B_3| = 2 \)
  - 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.cpp</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

Output format:

<Source file name> <source file version> <fault count>

MDS_Fault_count/MDS_Rep.source.Mds.Fw.Car.c++.CGIRegistration.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++.ComponentConnectorLinkInstance.cpp 1 0

SAS'02
References and Further Reading


References and Further Reading (cont’d)


