Toward a Quantifiable Definition of Software Faults

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

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

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).
Agenda

- Motivation
- Current State of Affairs
- Approach
  - Examples
- Identifying and Counting Faults
  - Examples
- Current Work
- Summary
- Future Work
Our Goals

- Understand fault insertion process
- Measure faults in evolving systems
- Understand the relationship between faults and structural code elements
- Developing software fault models depends on definition of what constitutes a fault
Measuring Faults

- 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
Measurement of Structural Evolution

Graph of Code Churn and Code Delta for the project ANONYMOUS
Structural Evolution at the Module Level

(Non-zero) Modules for build 2002-04-02 of project ANONYMOUS, sorted by Churn since baseline.
Measurement Framework

Fault Measurement and Identification

Proportional Fault Burden

Measurement

Fault Indices

Measurement Baseline

Compute fault index

Fault Indices

Place fault indices into repository

Measurement Repository

Compute Proportional Fault Burden

Fault Burden

Compute fault content regression model

Regression coefficients

Compute absolute fault burden

Absolute Fault Burden

Measure most recently changed source files

Rad structural measurements

Add structural measurements to repository

module name, revision number, structural measurements

module name, revision number, fault index

module name, revision number, fault count

module names, revision numbers, fault indices

CM Library

Extract changed source files

Most recently changed source files

Measure most recently changed source files

Raw structural measurements

Add structural measurements to repository

module name, revision number, fault count

module name, revision number, fault index

module names, revision numbers, fault indices

Extract Faulty Source Files

Fault Identification and Counting Rules

Identify Faults

Discovered Faults

Find initial Fault Occurrence

Initial Fault Placement

Add fault placement to repository

Repaired File IDs

Compare Repairs to Faulty Files

Fault Regions

Repaired Source Files

Problem Reports

Identify Source Files Repaired

Extract Repaired Source Files

Add fault placement to repository

Compute fault measurement

Absolute Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden

Fault Burden
No Existing Definition of Fault in Measurable Terms

- IEEE Standards

- ODC

- Previous work (Annual Oregon Workshop on Software Metrics, May 11-13, 1997)

- Frankl, Hamlet, Littlewood, Stringini (IEEE TSE, vol. 24, no. 8, August 1998)
Approach

- Examine changes made in response to reported failures
- Base recognition/enumeration of software faults on the grammar of the software system’s language
  - Faults found in executable, non-executable statements
- Fault measurement granularity in terms of tokens that have changed
Example 1

- Original statement: \( a = b + c \times d; \)
- Intended statement: \( a = b + c / d; \)
- One token changed – “\(*\)” \(\Rightarrow\) “/”
  - Coding error
- Count number of faults as 1
Example 2

- Original statement: \( a = b + c \times d; \)
- Intended statement: \( a = b + (c \times x) + \sin(z); \)
- Substantial difference between first and second statements
  - Reflects design rather than coding problem
  - Fault measurement method should reflect the degree of change
Identifying and Counting Faults

- Each line of text in each module version is 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
  - Modified in response to failures
- Use version control system to distinguish between changes due to
  - Repair
  - Functionality enhancements and other non-repair changes
Fault Identification

Example 1

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

- Original statement: \( a = b - c; \)
  - \( B_2 = \{<a>, \langle=\rangle, <b>, \langle-\rangle, <c>\} \)
- Modified statement: \( a = c - b; \)
  - \( B_3 = \{<a>, \langle=\rangle, <c>, \langle-\rangle, <b>\} \)
- \( B_2 - B_3 = \{ \} \)
- \( |B_2| = |B_3|, |B_2 - B_3| = 0 \)
- 1 fault representing incorrect sequencing
Fault Identification
Example 3

- 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
Current Work

- Current Work
  - Application to JPL software development effort
    - Research
    - Production
  - Develop better models relating
    - Structural measurements of software evolution during development
    - Number and types of faults inserted
Current Work (cont'd)

Faults vs. Cumulative Churn By Module
Smoothened Fault Counts Used, Last 2 Observations Removed

\[ y = 0.1574x + 1.13 \]

\[ R^2 = 0.7735 \]

Fault Model Example
Current Work (cont’d)

- Identifying fault insertion points
  - Increase resolution, accuracy of fault models
  - Determine empirical distribution of number of faults inserted per unit change
  - Supported by CM tool currently used in repository, CVS
Current Work (cont’d)

1.28 (jim 21-Mar-01): int watcher_parse_wrench_options (char * args)
1.28 (jim 21-Mar-01): {
1.28 (jim 21-Mar-01): RListOfRBuf * options; // The separated options.
1.28 (jim 21-Mar-01): RListOfRBufEntry * option_entry; // An entry in the list of options.
1.28 (jim 21-Mar-01): RListOfRBufEntry * option_parts; // Option entry broken into name and value.
1.28 (jim 21-Mar-01): RListOfRBufEntry * option_name; // The Name of the current option.
1.28 (jim 21-Mar-01): RListOfRBufEntry * option_val; // The Value (if applicable) of the current option.
1.28 (jim 21-Mar-01): ENode * command_line; // Node containing command line options.
1.28 (jim 21-Mar-01): int well_formed = TRUE; // Is this set of options well-formed?
1.28 (jim 21-Mar-01): }:
1.28 (jim 21-Mar-01):
1.28 (jim 21-Mar-01):
1.28 (jim 21-Mar-01):
1.31 (matt 04-May-01):
1.28 (jim 21-Mar-01): options = rstt_split (args, ":", 0):
1.28 (jim 21-Mar-01): command_line = watcher_config_get_command_line ()
1.28 (jim 21-Mar-01):
1.28 (jim 21-Mar-01): RLIST FOREACH (options, option_entry)
1.28 (jim 21-Mar-01):
1.28 (jim 21-Mar-01):
1.35 (coy 23-Jan-02):
1.28 (jim 21-Mar-01):
1.28 (jim 21-Mar-01):
1.28 (jim 21-Mar-01):
1.31 (matt 04-May-01):
1.28 (jim 21-Mar-01):
1.28 (jim 21-Mar-01):
1.28 (jim 21-Mar-01):
1.29 (jim 23-Mar-01):
1.28 (jim 21-Mar-01):
1.28 (jim 21-Mar-01):
1.28 (jim 21-Mar-01):
1.31 (matt 04-May-01):
1.28 (jim 21-Mar-01): if (! rbuf_equal_str (option_name->buf, "allow-nodrivers"))
1.28 (jim 21-Mar-01): {
1.28 (jim 21-Mar-01): /* Wrench will want to be able to start watcher up before any drivers are installed. */
1.28 (jim 21-Mar-01): }
1.29 (jim 23-Mar-01):
1.28 (jim 21-Mar-01): else if (! rbuf_equal_str (option_name->buf, "only-transport"))
1.28 (jim 21-Mar-01): {
1.28 (jim 21-Mar-01): /* Wrench wants to start watcher with only the specified transport */
1.31 (matt 04-May-01): /* (ie, most likely a unix socket or something, to send it commands without trying to connect to any GUIs) */
1.31 (matt 04-May-01): }
1.28 (jim 21-Mar-01): if (NULL == option_val)
1.28 (jim 21-Mar-01): {
1.29 (jim 23-Mar-01): printf("only-transport requires a value\n");
1.31 (matt 04-May-01): well_formed = FALSE;
1.28 (jim 21-Mar-01): break;
1.28 (jim 21-Mar-01): }
1.29 (jim 23-Mar-01):
1.28 (jim 21-Mar-01): }
1.29 (jim 23-Mar-01)
1.28 (jim 21-Mar-01): }
1.28 (jim 21-Mar-01): }
1.28 (jim 21-Mar-01): }
1.28 (jim 21-Mar-01): }
1.28 (jim 21-Mar-01): }
1.28 (jim 21-Mar-01): }
1.28 (jim 21-Mar-01): }
1.28 (jim 21-Mar-01): }
Measurement Noise

- Not all changes associated with a PR may actually be repairs
- "Pocket PRs"
  - Not an issue for this development effort because of how CM is set up
  - May be issue for other efforts
- Unequal test coverage – some components may be more heavily tested, finding more faults
Determination of Fault Insertion Point

- Line deletion, modification - relatively straightforward insertion point identification
  - CVS “annotate” command to find first version in which faulty line was inserted.

- Line addition – more difficult to determine fault insertion point
  - Approximation – identify insertion points of lines on either side of new line.
Inaccurate Fault Counting in Some Situations

- Example 1 – adding operators/operands
  - Original faulty statement: \( a = b + c; \)
  - Repaired statement: \( a = b - c + d; \)
  - Bag difference: \{\langle-\rangle, \langle d\rangle\}
  - 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
No information leakage from before the Big Bang
Summary

- Developed a starting point for measuring faults
  - Repeatable, consistent measurement
  - Faults measured at same level at which structural measurements are taken, i.e., function and method level
  - Easily automated
  - Transparent to developers
    - No additional activities for developers
    - No footprint in development environment
Future Work

- More accurate counts of number of tokens that have changed, reordered
- Extend fault type categories
- Extend technique to other software artifacts