Extending the Use of Measurement

Allen P. Nikora
Quality Assurance Office
Jet Propulsion Laboratory
California Institute of Technology

The work described in this presentation was carried out at the Jet Propulsion Laboratory, California Institute of Technology. Portions of the work were 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
- Measurement Goals
- Current Quality Measurement Techniques
  - Classical Software Reliability Models
  - RADC Fault Model
  - Phase-Based Model
  - COQUALMO
  - Estimating Fault Insertion Rates
  - Requirements Changes vs. Reliability
- Open Questions
- Where Do We Go Next?
- References and Further Reading
Overview

- Measurement is central to any engineering process
- All software design decisions governed by measurable outcomes
- All code development controlled by measurable outcomes
- All software test activity controlled by measures of test activity
- All software quality decisions quantified
- What can we do now, and what do we want to do but cannot?
Measurement Goals

- Monitor reliability of implemented system during test:
  - What is current system reliability?
  - How many resources will be needed to achieve required reliability?
  - What will the impact be if there are insufficient resources?
- Discriminate between fault-prone and non-faulty components prior to test
- Estimate/predict fault content of artifacts prior to test
Current Quality Measurement Techniques

Classical Software Reliability Modeling

• Statistical models used to estimate/predict reliability during test.
  - Input: Failure history (times between successive failures, number of failures observed in test interval of given length)
  - Output: pdf for interfailure time, number of failures in test interval

• Used to monitor reliability change (hopefully growth) during software test.

• Over 100 models published to date – details of most popular models given in [Lyu96].

• 10 popular models implemented in CASRE – software reliability estimation tool available at http://www.openchannelfoundation.org/projects/CASRE_3.0
Current Quality Measurement Techniques (cont’d)
Classical Software Reliability Modeling

Region 1 - debugging, infant mortality
Region 2 - useful life period
Region 3 - component wear-out or fatigue

Hazard rate z(t)

Time t
Current Quality Measurement Techniques (cont’d)

Classical Software Reliability Modeling – CASRE Output – Interfailure Times

SEHAS’03
Current Quality Measurement Techniques (cont’d)
Classical Software Reliability Modeling – CASRE Output – Failure Intensity

SEHAS'03
Classical Software Reliability Modeling

- Limitations
  - Model assumptions may not match test process
  - Required data may not be available
  - May not be possible to use model results to improve current software design or development process:
    - Modeling takes place after the design and coding phases have been completed.
    - Model results cannot be readily related to product or process characteristics.
Current Quality Measurement Techniques (cont’d)

RADC Fault Model

- Study of 59 software development efforts, sponsored by Rome Air Development Center in mid 1980s
- Purpose - develop a method for predicting software reliability in the life cycle phases prior to test. Acceptable model forms were:
  - measures leading directly to reliability/failure rate predictions
  - predictions that could be translated to failure rates (e.g., error density)
- Details given in [McCal87]
Current Quality Measurement Techniques (cont’d)

RADC Fault Model

- Error density at the start of test given by:

\[ \partial_0 = A \times D \times (SA \times ST \times SQ) \times (SL \times SS \times SM \times SU \times SX \times SR) \]

- A -- Application Type (e.g. real-time control system, scientific computation system, information management system)

- D -- Development Environment (characterized by development methodology and available tools). The types of development environments considered are the organic, semi-detached, and embedded modes, familiar from the COCOMO cost model.
Current Quality Measurement Techniques (cont’d)

RADC Fault Model

- Requirements and Design Representation Metrics
  - SA - Anomaly Management
  - ST - Traceability
  - SQ - Incorporation of Quality Review results into the software

- Software Implementation Metrics
  - SL - Language Type (e.g. assembly, high-order language, fourth generation language)
  - SS - Program Size
  - SM - Modularity
  - SU - Extent of Reuse
  - SX - Complexity
  - SR - Incorporation of Standards Review results into the software
Current Quality Measurement Techniques (cont'd)
Phase-Based Model

- Developed by John Gaffney, Jr. and Charles F. Davis of the Software Productivity Consortium. Details given in [Gaff88].
- Makes use of error statistics obtained during technical review of requirements, design and the implementation to predict software reliability during test and operations.
- Can also use failure data during testing to estimate reliability.
- Assumptions:
  - Development resources applied according to a Rayleigh curve.
  - The development effort's current staffing level is directly related to the number of errors discovered during a development phase.
  - The error discovery curve is monomodal.
  - Code size estimates are available during early phases of a development effort.
  - Fagan inspections are used during all development phases.
Current Quality Measurement Techniques (cont'd)

Phase-Based Model

- Number of errors discovered during a life cycle phase:

\[ \Delta V_t = E\left( e^{-B(t-1)^2} - e^{-Bt^2} \right) \]

- "t" is a life cycle phase index:
  - t = 1 - Requirements Analysis
  - t = 2 - Software Design
  - t = 3 - Implementation
  - t = 4 - Unit Test
  - t = 5 - Software Integration Test
  - t = 6 - System Test
  - t = 7 - Acceptance Test

SEHAS'03
Current Quality Measurement Techniques (cont’d)

Phase-Based Model

Typical Error Discovery Profile

Error Density (Errors per KLOC)

Development Phase Index

SEHAS'03
Current Quality Measurement Techniques (cont’d)

COQUALMO

- Under development at University of Southern California Center for Software Engineering. More information available at [Chu99], http://sunset.usc.edu/research/coqualmo/index.html.
- Goal – understand tradeoffs between cost, schedule, and quality
  - Can be used for predicting number of residual defects/KSLOC (Thousands of Source Lines of Code) or defects/FP (Function Point) in a software product.
  - Can be applied in the early activities such as analysis and design, as well as in the later stages for refining the estimate when more information is available.
  - Enables 'what-if' analyses that demonstrate the impact of various defect removal techniques and the effects of personnel, project, product and platform characteristics on software quality. It also provides insights into determining ship time, assessment of payoffs for quality investments and understanding of interactions amongst quality strategies.
Current Quality Measurement Techniques (cont’d)

COQUALMO Model Framework

Defect Introduction Pipes

Design Defects

Requirements Defects

Coding Defects

Defect Removal Pipes

Residual Software Defects

SEHAS'03
Current Quality Measurement Techniques (cont’d)

Estimating Fault Insertion Rates

- Developed by Munson and Nikora. Details in [Mun02, Nik01, Nik03, Nik03a].

- Developed to
  - Understand fault insertion mechanism in evolving systems
  - Estimate/predict fault content for evolving software systems

- Findings to date
  - Number of faults inserted is proportional to total amount of measured change
  - Some types of changes are more likely to result in fault insertion than others.
Current Quality Measurement Techniques (cont’d)

Requirements Changes vs. Reliability

• Developed by Schneidewind - details in [Schn02]
• Goal – understand how measurable changes to requirements affect the quality of the implemented system.
• Analyzed 2 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

SEHAS’03
Current Quality Measurement Techniques (cont'd)

Requirements Changes vs. Reliability

\[ CF = 6 \times 10^{-7} \times CS^2 - 0.0003 \times CS + 1.9511 \]

Cumulative Failures vs. Cumulative Memory Space

SEHAS'03
Current Quality Measurement Techniques (cont'd)

Requirements Changes vs. Reliability

Cumulative Failures vs. Cumulative Issues

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

CF: Cumulative Failures
CI: Cumulative Issues
Open Questions

- How can we extend to earlier life cycle phases current techniques for measuring artifacts?
- Does it make sense to measure formal specifications in attempting to predict faults?
- How does the development process affect software quality?
- Can we model the cognitive process leading to the introduction of faults in artifacts?
Where Do We Go Next?

You tell me...
References and Further Reading


References and Further Reading (cont’d)


