Software Fault Content Prediction Model

Allen P. Nikora
University of Southern California
Center for Software Engineering

Affiliates’ Annual Research Review
March 10, 1997

Agenda

I: The Software Reliability Issue
II: Survey of Software Reliability Models
III: A New Model for Fault Discovery and Removal
IV: Status and Plans
Part I: The Software Reliability Issue

- Software reliability is often the major factor in system reliability.
- Current state of the art:
  - Estimate and predict software reliability during system execution.
  - Estimate software fault content during implementation and design phases based on structural characteristics of the system.
- Goal - use product and development process measures as reliability model inputs during earlier development phases.

Software component reliability is often the major determining factor in system reliability. For one project at JPL, an analysis of reliability which assumed software reliability was 1 led to an estimate of system's MTTF of 123 hours at the start of system test. Taking software reliability into account for this system led to an estimated MTTF of about 120 minutes at the start of system test.

- Current state of the art:
  - Estimate and predict software reliability during testing and operational phases. These techniques have the following benefits:
    - Determine when a reliability goal has been achieved
    - Determine release date for a software system
    - Better control over use of testing resources
    - Status monitoring during test phases
    - Quantitative risk evaluation
  - Estimate software fault content during implementation and design phases based on structural characteristics of the system. This type of model would allow model results to be used to guide changes to the software system's design to optimize reliability. However, model predictions are difficult to update and do not address development process characteristics very well.

- Goal - use product and development process characteristics as input to reliability model during earlier development phases. Hope is that model outputs could be used to guide product design and development process to optimize reliability.
Part II: Survey of Software Reliability Models

- Reliability prediction during test
  - Common Reliability Model Assumptions
  - Limitations

- Reliability prediction prior to test
  - Current Predictive Models
  - Limitations of Current Techniques
Reliability prediction during test

- **Limitations**

  - **Applicability of Assumptions** - Model assumptions may not match what happens during testing.
  - **Availability of required data** - Time between subsequent failures, which is a required input for many models, may not be collected. Usage pattern information (see next point) may also be unavailable.
  - **Operational reliability** may be different from that observed during test, since usage patterns may differ between test and operations. Even with good estimates of operational usage, it is still possible to inaccurately estimate reliability. AT&T has reported this type of experience in connection with upgrades to their equipment.
  - **Current techniques** are applicable only during the testing phases. Model results cannot be used to improve software design or the development process for the current development effort, since:
    - Modeling takes place after the design and coding phases have been completed.
    - Model results cannot be readily related to product or process characteristics.
Currently-available techniques
- Rome Air Development Center study
- Relative complexity
- Phase-Based model
- JPL empirical model

Limitations of Current Techniques
- Methods may use measures which are not easily obtained
- Development process is often incompletely characterized
- Model predictions are not easily updated

Limitations
- RADC study
  - Required measurements can be difficult to obtain during early development phases (e.g., module size, complexity)
  - Method does not take schedules and staffing profiles into account.
  - Method cannot handle situations in which development methods change during a project, or a project on which development practices are not consistently applied.
- Relative Complexity
  - Uses only complexity measures as reliability predictors.
  - Given current development practices, often cannot be applied prior to coding phase.
- Phase-Based Model
  - Does not use development process or product characteristics as predictors.
  - Cannot handle development efforts in which the products developed during each phase are not completely inspected.
  - May not perform well for projects whose staffing profiles aren’t close to a Rayleigh curve.
- JPL Empirical Model
  - Same basic limitations as the Phase-Based Model.
  - Does not assume a Rayleigh curve staffing profile, but there is an implicit assumption that the typical JPL staffing profile is followed. This may not hold for other organizations.
Part III: A New Model for Fault Discovery and Removal

- Desired model characteristics
- General model formulation
- Advantages of model

Desired model characteristics:
- Relate characteristics of product and development process to software reliability.
- Make use of measures that are available during each development cycle.
- Predictions should be easily updatable to account for changes in the development method or the system being developed.
- Results should be expressed in a form usable to developers, customers, and users. Ideally, the model output would be a hazard rate that could be used to compute reliability.
Factors influencing the number of faults in a software system:

- **Software system characteristics:**
  - Complexity - different measures may be used for different phases:
    - Number of statements, number of nodes, Halstead measures as input to relative complexity analysis during detailed design, implementation
    - Function points or object points during architectural design
    - COCOMO II classifications during requirements specification
  - Traceability
  - Extent of reuse

- **Development process characteristics:**
  - Development environment, per the COCOMO II classifications:
    - Number of development personnel
    - Experience level of developers
      - Amount of experience with application type
      - Amount of experience with development method
    - CMM Maturity level of development organization
  - Current development phase (e.g., requirements specification, architectural design)
  - Use of Fagan inspections or other technical reviews:
    - Fraction of product inspected
    - Amount of time spent in preparation
    - Product volume inspected per meeting
    - Number of inspectors
    - Amount of time spent in meeting
  - Number of faults already in the product.
The rate matrix $A$ can be filled in by observing software development efforts in progress and measuring the time between the appearance and removal of successive faults.

From the rate matrix $A$, the probability transition matrix $P$ is computed as $P = \exp(At)$, where $T$ is the length of time over which we're interested in computing the number of remaining errors.

Given $P$, the probability density function for the number of errors remaining at the end of phase $i$ is given by:

$$ r_i = P(t_i, r_{i+1}) $$

where

- $r_i$ is the output pdf for the current phase, $i$.
- $P(t_i)$ is probability transition matrix for phase $i$ which is expected to have a duration of $t_i$.
- $r_{i+1}$ is the input pdf for phase $i$, which is the output pdf of phase $i-1$. 
Part III: A New Model for Fault Discovery and Removal (cont’d)

Advantages of the new model:
- Ability to compute confidence values.
- Ability to make cost/quality tradeoffs earlier in the development effort.
- Ability to refine and update predictions as more detailed information about product and process becomes available.
- Predictions are in terms meaningful to users and developers.

- Ability to compute confidence values for number of faults remaining at the end of each development phase.
- Given a choice between development methods and product structure, software managers would be able to make more informed trade-offs between cost, schedule, and quality earlier in the development cycle.
- Ability to refine and update predictions as more detailed information about product and process becomes available. In particular, it becomes straightforward to accommodate changes to development process or the system being developed. For example, the development environment may change during the design phase, say at time \( t_1 \). We would compute the pdf for the number of faults remaining at time \( t_1 \) from project inception at \( t_0 \). The new information concerning the development environment after \( t_1 \) would then be used to compute the pdf for the number of faults remaining at the end of the development effort. Existing methods, such as the JPL empirical model and the Phase-Based model, cannot accommodate such changes.

The number of errors expected at time \( t_1 \) is represented by:

\[ r_{t_1} = P(t_1 | t_0) i_{t_0} \]

The number of errors expected by the end of the development phase is:

\[ r_{t_f} = P(t_f - t_i | t_i) r_{t_i} \]

- Predictions are in terms meaningful to users and developers. Developers are concerned with the number of faults that will be seen in later development phases. Both users and developers are concerned with number of faults that will be seen during operations.
Part IV: Status and Plans

- Status
  - Scope of Work
  - Data Collection
  - Analysis Methods
- Limitations of Model
- Future Work
Status

- Scope of work
  - Original intent was to study all phases of development efforts.
  - Current focus is analysis of implementation phase and products. This is because:
    - Data is not consistently available.
    - Relative ease of source code measurement.
    - Ease of measuring code evolution.
  - Preliminary form of model developed; data collection for calibration in progress.
  - Current work expected to lead to improved methods of studying earlier phases.

Status

- Scope of work
  - Original intent was to study all phases of development efforts at JPL, other NASA centers, and selected military systems.
  - Current focus is determining rates of fault introduction and removal from product and process characteristics during the implementation phase. This is because:
    - The data required for a study of all development phases is not consistently available among development efforts. For instance, Fagan inspection data may be available for only a portion of a development phase.
    - The products developed during the implementation phase are easily measurable, while those produced during requirements and design phases often cannot be measured (e.g. design documentation in a mixture of natural language, state diagrams, and data flow diagrams in forms that aren't machine readable).
    - The evolution of the source code is relatively easy to measure and correlate with the fault content at any particular time. This is because of the extensive use of version control tools such as SCCS and RCS which makes establishment of versions and deltas within versions a straightforward task. This is not necessarily the case during requirements and design phases.
  - It is expected that results of current work will lead to improved methods for studying phases prior to implementation.
Data collection

- Current data collection efforts are focused on characterizing the development process and the products during the implementation phase. The following information is being collected from JPL, other NASA, and military development efforts:
  - COCOMO II questionnaires are given to software managers and software quality assurance leads for each effort. The researcher assists these individuals in completing the form by providing clarification in uncertain areas.
  - Software developers provide the contents of the revision control systems to the researcher. This allows the researcher to measure the structural aspects of the code with minimal impact to the developers.
  - Institutional problem reporting systems provide information on software errors discovered during development test and system-level integration.
  - Effort profiles during implementation are provided by the project office.

- Data Collection Issues
  - Problem reporting information can be incomplete. To locate a fault to the function/procedure level, all of the following information must be analyzed:
    - Memos attached to problem reports detailing corrective action. These may be available from the developers.
    - Header information in the revision control system files. The headers may contain detailed information about which function(s) were altered to correct a particular problem.
    - Differences between deltas of a specific module in which a repair was made. If it is known that a repair was made for delta “x”, a differential comparison can be made between deltas “x” and “x-1” to isolate the portions of the module which changed. These changes can then be associated with the appropriate function(s).
  - Problem reporting information does not identify the point at which the problem first appeared in the software. To determine when the problem first appeared, it is necessary to examine all of the deltas prior to the delta in which the problem was repaired to see when the problem first appeared.
Analysis methods

- Structural characteristics of the software system are measured by relative complexity (see Additional Details). This measurement technique is extended to measure the evolution of the system, both from version to version, and from delta to delta within any particular version. “Code delta” measures the change in relative complexity from version to version: if code is added, code delta is positive, and if code is taken away, code delta is negative. Code churn measures the absolute value of the change in relative complexity from version to version.

- Problem reporting information is analyzed to determine the delta in which a fault first appeared within the software system.

- Correlation between rate of fault appearance and rate of system’s evolution is calculated to establish relationship between changes in the system’s structure and the number of faults present in the system.

- Multivariable regression analysis is used to establish relationships between development process characteristics, structural characteristics, and the rate of fault introduction and removal.
Limitations of Model

- Model calibration information for early development phases is more difficult to obtain.
- The model does not produce predictions in a form related to reliability.
- The model may be computationally difficult.

Limitations of Model

- Information required to calibrate the model may be more difficult to obtain for early development phases than for later ones. This is because products developed during early phases can be difficult to measure, and it is often difficult to identify distinct versions in order to measure the evolution of the system.
- The model does not produce predictions in a form related to reliability. Rather, it produces predictions in terms of fault content. If the operational profile of the system is known or can be estimated, then it would be possible to extend the model to produce reliability related estimates. However, this cannot be done purely using only static product and development process characteristics.
- The computations involved may be difficult. The Fokker-Plank equations specified in the rate matrix may not have analytic solutions, and the size of the rate matrix has no definite upper bound. It can become necessary to employ numerical methods to solve the Fokker-Plank equations. The size of the rate matrix can be estimated by establishing a threshold - for instance, the size of the rate matrix may be doubled as long as the difference between the 95% confidence value for the number of faults for successive iterations exceeds that threshold. Finally, for real systems, the rate matrix can become very large.
Future work

- Acquire more data about implementation phase. Additional information will be used to establish more accurate and complete relationships between development process characteristics and the rate of fault introduction and removal.

- Study earlier development phases for participating efforts. Methods used in studying the implementation phase should, in principle, be applicable to earlier phases. It will be necessary to solve the data collection issues mentioned earlier.

- Identify methods of relating measures of fault count to reliability. Current work by Munson at the University of Idaho on “functional complexity” could be used as a basis for this work. The key here is to somehow be able to estimate operational profiles during the implementation and earlier phases of a development effort.
Additional Details

- Relative Complexity
Relative Complexity

Relative complexity is the name given to a technique used to isolate the sources of variation underlying the set of software complexity metrics describing a software system.

Principal components analysis is used to accomplish this isolation. In principal components analysis, we seek to account for most of the variability in the set of attributes of the matrix with just linear combinations of them attributes. An example is given below:

Consider a set of \( n \) program modules, each module having \( m \) software attribute measures. This information is represented as an \( nxm \) matrix. The covariance matrix for this information is represented by \( \Sigma \).

\( \Sigma \) can be decomposed as:

\[
\Sigma = \mathbf{T} \Lambda \mathbf{T}^T
\]

where \( \Lambda \) is the diagonal matrix of eigenvalues,

\[
\sum_{i=1}^{m} \lambda_i = \text{tr}(\Sigma)
\]

\( \mathbf{T} \) is an orthogonal matrix whose columns \( j \) are the eigenvectors associated with \( \lambda_j \), and \( \mathbf{T}^T \) is the transpose of \( \mathbf{T} \).

Principal components analysis (cont'd):

The \( m \) eigenvectors in \( \mathbf{T} \) give the coefficients that define the uncorrelated linear combinations of the original complexity metrics.

The ratio \( \lambda_j/\text{tr}(\Sigma) \) gives the proportion of complexity metric variance that is explained by the \( j \)th principal component.

The first few principal components typically explain a large proportion of the total variance. A typical stopping rule selects principal components with associated eigenvalues greater than one.

The standardized transformation matrix \( \mathbf{T}^* \) is then constructed from \( \mathbf{T} \) to produce \( p \) domain metrics for each of the \( n \) programs in the system. An element \( t_{ij} \) of \( \mathbf{T}^* \) gives the weight of the \( i \)th complexity metric, \( i=1,2,\ldots,m \), for the \( j \)th domain metric, \( j=1,2,\ldots,p \).

Given \( \mathbf{x} \), the vector of standardized metrics for program \( k \), \( \mathbf{d}_k = \mathbf{x}_k^T \mathbf{T}^* \) is the new vector of domain metrics for the \( k \)th program module.
Relative Complexity (cont'd)

- Relative complexity is computed by forming the weighted sum of the domain metrics:
  \[ \rho_i = d_k \Lambda^* \]
  where \( \Lambda^* \) is the vector of eigenvalues associated with the selected domain,
  \( \Lambda^{**} \) is the transpose of the vector, and
  \( d_k \) is the vector of domain metrics for program \( k \).

- Values of \( \rho \) are distributed with mean 0 and variance
  \[ \mathcal{V}(\rho) = \sum_{i=1}^{n} \lambda_i \]

- For ease of interpretation, relative complexity is often scaled as follows:
  \[ \rho_i' = \frac{-10 \rho_i}{\sqrt{\mathcal{V}(\rho)}} + 50 \]

This metric has a mean of 50 and standard deviation of 10.

USC Center for Software Engineering
3/10/97
slide 21