

# Developing Fault Models for Space Mission Software

Allen P. Nikora  
Jet Propulsion Laboratory,  
California Institute of Technology  
Pasadena, CA 91109-8099  
Allen.P.Nikora@jpl.nasa.gov

John C. Munson  
Computer Science Department  
University of Idaho  
Moscow, ID 83844-1010  
jmunson@cs.uidaho.edu

## Abstract

*Over the past several years, we have focused on developing fault models for space mission software. In general, these models use measurable attributes of a software system and its development process to estimate the number of faults inserted into the system during its development; their outputs can be used to better estimate the resources to be allocated to fault identification and removal for all system components. Working with the Mission Data System at JPL, we have identified relationships between the amount of structural change made to a software system its development and the number of faults inserted into it. To develop fault models, practical mechanisms for measuring a software system's structural change and the number of faults inserted into it must be implemented. We discuss the required attributes of these mechanisms and their implementation at JPL.*

## 1. Introduction

Over the past several years, a great deal of work has been done in the area of using measurements of software systems to identify fault-prone components and predict their fault content. This would allow software developers to more accurately identify fault-prone components of the system, estimate the number of faults inserted into a software system at various points during its development, and estimate the resources that would need to be applied to the fault identification and removal efforts for specific system components. Examples of this work include the classification methods proposed by Khoshgoftaar and Allen [1] and by Ghokale and Lyu [2], Schneidewind's work on Boolean Discriminant Functions [3], Khoshgoftaar's application of zero-inflated Poisson regression to predicting software fault content [4], and Schneidewind's investigation of logistic regression as a discriminant of software quality [5]. Each of these efforts has provided useful insights into the problem of identify-

ing fault-prone software components and estimating software components' fault burden prior to test.

However, all of these efforts analyzed a snapshot of the subject system, rather than examining its evolution during development. This may have limited the validity of those efforts' conclusions to the point in the development life cycle when the measurements were made. If, however, the entire evolution of a software system were to be analyzed, the conclusions should be applicable to any point in the development cycle of the artifact being studied. With this goal in mind, we conducted a small study on of the CASSINI flight software several years ago [6, 7]. During this study, we discovered that the number of faults inserted into that system during its development was directly proportional to the measured amount of structural change. However, there were two limitations to this study:

- The study was relatively small – fewer than 50 observations were used in the regression analysis relating the number of faults inserted to the amount of structural change.
- The definition of faults that was used was not quantitative. The ad-hoc taxonomy, first described in [8], was an attempt to provide an unambiguous set of rules for identifying and counting faults. The rules were based on the types of changes made to source code in response to failures reported in the system. Although the rules provided a way of classifying the faults by type, and attempted to address faults at the level of individual functions or methods (modules), they were not sufficient to enable repeatable and consistent fault counts by different observers to be made. The rules in and of themselves were unreliable.

We started working in collaboration with the Mission Data System (MDS) at JPL [9] in the fall of 2000 with the goals of:

- Overcoming the limitations of the CASSINI study, and

- Developing practical software measurement techniques that could be infused into space mission software development efforts at JPL and other NASA centers to improve developers' ability to control software quality.

The amount of data available from the MDS overcame the first limitation of the earlier study of CASSINI. We measured the structural evolution of the MDS and the number of faults repaired over a period beginning on October 20, 2000, and ending on April 26, 2002. Over that time, we counted over 15000 distinct modules and more than 1500 builds of the MDS. The total number of distinct versions of all modules was greater than 65,000. Over 1400 problem reports were included in the analysis; these problem reports provided the information from which the number of repaired faults was computed. By developing a quantitative definition of software faults [10], we were able to overcome the second limitation of the earlier study.

Our analysis of the measurements taken of these MDS artifacts confirmed the results of the earlier study by showing statistically significant relationships between the measured amount of structural change a software system undergoes during its development and the number of faults inserted into that system [12]. Briefly, the number of faults inserted into a software component once it has been created is directly proportional to the amount of structural change made to that component over its lifetime - over 60% of the variation in the number of faults repaired was explained by the measurements of the system's structural evolution. We were also able to extend the results of the earlier study by differentiating between changes that would be more likely to result in faults and changes that would be less likely to do so [13]. We are now starting to infuse the measurement techniques developed during this study into other JPL and NASA software development efforts. The remainder of the paper discusses the required characteristics of the measurement mechanisms, their current implementation at JPL, and our plans for future work.

## 2. Measurement Mechanisms for Developing Fault Models

To develop fault models, mechanisms for measuring a software system's structural and development process characteristics must be implemented, as must mechanisms for measuring the number of faults inserted into the system. Fault models are developed using the measurements of the system's structure and its development process as independent variables, and the number of faults discovered as the dependent variable. Our experience indicates that these mechanisms must have the following characteristics:

- **The mechanisms must provide repeatable and accurate measurements** of the system's structural characteristics, its development process, and the number of faults inserted into that system. Measuring the structural attributes of a software system is a straightforward activity; numerous measurement tools are available to take the required measurements. Measuring the number of faults inserted into the system is a more complicated problem, requiring a precise notion of what constitutes a software fault. The development process may be characterized in several ways. For instance, the development process may be characterized in the same way as the COCOMO and COCOMO II cost models [14, 15]. We could also determine the extent to which a development process is consistent with a given set of Key Process Areas of the Software Engineering Institute's Capability Maturity Model [16, 17], or use a classification scheme similar to that described by McCall et al. [18].
- **Structural attributes, development process characteristics, and the number of faults inserted should be measured at the same level of detail.** If we measure structural characteristics of a software system at the level of individual modules, we should also measure the number of inserted faults and development process characteristics at that same level in order to develop fault models that can be used to estimate the fault content of individual modules.
- **The perceived benefit of making these measurements must outweigh the perceived costs of the measurement activity.** In practical terms, this usually means that developers should not be required to expend any effort in obtaining the required measurements. This can be achieved by automating the measurement process and making it invisible to the development teams (i.e., the measurement process must not have any noticeable effect on the development environment). In our experience, software developers have little or no experience with static software measurement. If they are confronted with the measurement task, they will simply fail to do it.

## 3. A Measurement System Implementation

As part of our collaboration with the MDS, we have implemented and installed a system to obtain and manage the structural and fault measurements required to develop fault models [19, 20]. Since the system is designed to manage measurements of software evolution, we have named it Darwin. Darwin is a software engineering management system for tracking the progress of evolving

software systems. It is a repository for all of the engineering data surrounding software development and software test. It is a web server that permits developers and testers to interact with the system. It is an analytical tool that provides the measurement capabilities for both software development and software test. All management aspects of the software development and testing process are to be maintained by the Darwin system.

The Darwin software system resides in a network appliance computer (NA) attached to the local intranet connecting systems analysts, software developers, software testers, software managers, and the software quality staff. The measurement database, the metric tools, and all of the ancillary support software for Darwin are all serviced from this machine. To interact with the system a user will visit a web site on the server and will then be linked through web pages provided by the appliance to the appropriate application. This network appliance approach greatly simplifies the problem of instituting a measurement program in that it obviates the need to port the software measurement infrastructure from platform to another.

Of greatest importance to the whole development process is the cogency of the information. A development manager can see the status of the development process essentially in real time. Specific reports on change activity can assist in the process of setting milestones and reviews. A test manager can evaluate the status of a test process, again in near real time. The NA is a visible aid to developers, managers, and testers. All of the engineering data about the evolving software is managed from this location.

On the static measurement side, Darwin is designed to track changes to code and faults reported against specific code modules. To measure specific changes to the source code system can be a very complex process in that a typical large software system may have many developers working with the same code base at the same time. It is possible to track changes to code at the individual version level. Experience has shown that this kind of resolution is seldom, if at all, necessary. What is of primary interest to developers and managers is the code that constitutes a build.

At the point of a new build, a new build index vector will be developed by a build manager. This build index vector will determine which source code elements actually go to the build. On completion of the build index vector, Darwin can be requested to measure the source code for the build. Currently, the Darwin NA can supply measurement tools for C and C++ with our Extended C Metric Analyzer (ECMA) or for Java with our Java Metric Analyzer (JMA). The measurement tools are a component of the Darwin NA.

With the contents of the build index vector, Darwin will systematically retrieve from the configuration control system the appropriate source code elements, with the updated versions of this code. Each source code module will then be measured by the appropriate measurement tool and the raw metrics for that module will then be incorporated into the database. It is important to observe that the only modules that are measured by the system are those that will result in executable code. All header files and compiler directives will have been resolved before the measurement process can begin.

Just as important as the process of measuring code is the process of maintaining specific fault reports. Each fault report will report on exactly one fault on one code module. These fault reports are served from the NA as web pages and are tracked by Darwin. For each change to a system, there can be exactly two reasons for the change. A fault has occurred or there has been a change in the program specifications. All program changes relative to faults are tracked by Darwin.

The Darwin system is equipped with a set of standard SQL generated reports that reflect the more typical database inquiries that will be made by most managers. Darwin tracks changes in source code across builds. One way of characterizing software systems for reporting purposes is in terms of changes to the structure of the code in the attribute domains [11, 19]. We obtain the attribute domain scores by performing a principal component analysis (PCA) on the standardized raw measurements [21]. We do this to identify the distinct orthogonal sources of variation and map the raw metrics onto a set of uncorrelated metrics that represent essentially the same information contained in the original metrics – although we may take a dozen different measurements of the system, there will not necessarily be a dozen distinct sources of variation. Software systems can also be characterized in terms of the fault burden of the system as measured by weighted sums of the attribute domains - the system Fault Index, code churn and code deltas [11, 19]. This gives us considerable insight as to where existing faults might be and continuing processes that are potentially introducing new faults. These data are useful in and of themselves for prioritizing the software review processes. We can devote our software review team energies to those regions of the code where the greatest problems are likely to be.

Figure 1 and Figure 2 show examples of the types of reports available through the Darwin web interface. Figure 1 shows a system-level plot of software evolution. Each point on the plot represents an individual build of the system. The ordinate represents the build date, while the abscissa represents the cumulative amount of change to the system. The system-level cumulative change is computed by summing the cumulative amount of change



pository from which it can obtain measurements. In the case of the MDS, this is nothing more than creating a copy of the configuration management repository that can be transferred to the NA. Developers need not be burdened even with this task; our experience with MDS indicates that this is a task that can be easily performed by members of a project's software quality staff. Since there are no measurement or reporting mechanisms implemented in the development environment, a development organization will not experience any adverse effects related to a failure of these mechanisms.

#### 4. Discussion and Future Work

To obtain the types of measurements required to develop fault models for software systems, we have developed Darwin, a software engineering management system, and deployed it in collaboration with the MDS development effort. Currently, we are using Darwin to obtain static measurements only; we have succeeded in measuring MDS structural evolution and fault repair counts at the module level over an 18-month period. We have used this information to confirm with greater accuracy earlier work on relationships between measurements of structural change and the fault insertion rate; we have also been able to extend the earlier work by differentiating between types of change more likely to result in the insertion of faults and those types of change less likely to do so. Future work includes infusing Darwin's capability to track dynamic measurements - failure reports and test execution profiles - into the MDS and other collaborating JPL and NASA software development efforts. By combining these dynamic measurements with the static measurements of software evolution and fault repairs, it will become possible to:

- Estimate the fault-finding effectiveness of a given set of test cases and procedures, and
- Make estimates of a software system's risk of exposure to residual faults.

To date, we have made no significant use of measured development process characteristics in developing fault models. First of all, the development process characteristics we are able to measure apply to the entire system, rather than to individual modules. This means it would be necessary to analyze several projects in order to say anything meaningful about the effect of the development process on fault insertion rates. A rule of thumb states that in developing a model, one should have four observations for every parameter in that model. If we were to characterize the development process as described in [15], we would need to analyze approximately 80 different development efforts. At this point, we have

simply not accumulated the required number of observations. As part of our future work, we would like to examine in greater detail the effect of the development process on fault insertion rates. By infusing the Darwin network appliance into additional JPL and NASA development efforts, we hope to be able to obtain the required measurements.

As it stands, Darwin can be deployed for a wide variety of software development efforts. Since it is implemented as a network appliance apart from the development environment, measurement activities are invisible to the development team - there is no additional effort on their part required to perform the measurement activities, and any unexpected behavior of the measurement system will not adversely impact the development team. The web-based reporting mechanism allows development teams, testers, and managers to easily obtain the relevant information to make informed decisions about the software development process and the software quality/testing process.

#### Acknowledgments

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 IV&V Facility. The authors wish to thank the MDS project for the cooperation that made this study possible.

#### References

- [1] T. M. Khoshgoftaar, E. B. Allen, "Modeling Software Quality with Classification Trees", in H. Pham (ed), *Recent Advances in Reliability and Quality Engineering*, Chapter 15, pp 247-270, World Scientific Publishing, Singapore, 2001.
- [2] S. S. Gokhale, M. R. Lyu, "Regression Tree Modeling for the Prediction of Software Quality", proceedings of the Third ISSAT International Conference on Reliability and Quality in Design, pp 31-36, Anaheim, CA, March 12-14, 1997
- [3] N. F. Schneidewind, "Software Metrics Model for Integrating Quality Control and Prediction", proceedings of the 8th International Symposium on Software Reliability Engineering, pp 402-415, Albuquerque, NM, Nov, 1997.
- [4] T. Khoshgoftaar, "An Application of Zero-Inflated Poisson Regression for Software Fault Prediction", proceedings of the 12<sup>th</sup> International Symposium on Software Reliability Engineering, pp 66-73, Hong Kong, Nov, 2001.
- [5] N. F. Schneidewind, "Investigation of Logistic Regression as a Discriminant of Software Quality", proceedings of the 7th International Software Metrics Symposium, pp 328-337, London, April, 2001.

- [6] J. Munson and A. Nikora, "Estimating Rates Of Fault Insertion And Test Effectiveness In Software Systems," Proceedings of the Fourth ISSAT International Conference on Reliability and Quality in Design, August 12-14, 1998 pp. 263-269.
- [7] A. P. Nikora, J. C. Munson, "Determining Fault Insertion Rates For Evolving Software Systems", proceedings of the 1998 IEEE International Symposium of Software Reliability Engineering, Paderborn, Germany, November 1998, IEEE Computer Society Press.
- [8] A. Nikora, J. Munson, "Finding Fault with Faults: A Case Study", with J. Munson, proceedings of the Annual Oregon Workshop on Software Metrics, Coeur d'Alene, ID, May 11-13, 1997.
- [9] D. Dvorak, R. Rasmussen, G. Reeves, A. Sacks, "Software Architecture Themes In JPL's Mission Data System", AIAA Space Technology Conference and Exposition, September 28-30, 1999, Albuquerque, NM.
- [10] J. Munson, A. Nikora, "Toward a Quantifiable Definition of Software Faults", Proceedings of the 13<sup>th</sup> IEEE International Symposium on Software Reliability Engineering, IEEE Press.
- [11] J. Munson, Software Engineering Measurement, CRC Press, 2003.
- [12] A. Nikora, J. Munson, "Developing Fault Predictors for Evolving Software Systems", submitted to the 9<sup>th</sup> International Symposium on Software Metrics, September 3-5, 2003, Sydney, Australia
- [13] A. Nikora, J. Munson, "Understanding the Nature of Software Evolution", submitted to the International Conference on Software Maintenance, September 22-26, 2003, Amsterdam, The Netherlands
- [14] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Inc., 1981.
- [15] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, R. Selby, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," *Annals of Software Engineering*, volume 1, J.C. Baltzer Science Publishers, Amsterdam, The Netherlands, 1995, pp. 57-94.
- [16] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, Charles V. Weber, "Capability Maturity Model<sup>SM</sup> for Software, Version 1.1", Technical Report CMU/SEI-93-TR-024 ESC-TR-93-177 Feb 1993.
- [17] Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, Marilyn Bush, "Key Practices of the Capability Maturity Model<sup>SM</sup>, Version 1.1", Technical Report CMU/SEI-93-TR-025 ESC-TR-93-178, Feb, 1993.
- [18] J. McCall, J. Cavano, "Methodology for Software Reliability Prediction and Assessment," Rome Air Development Center (RADC) Technical Report RADC-TR-87-171. volumes 1 and 2, 1987
- [19] "Specifications For the Darwin Network Appliance", Cylant, March, 2002
- [20] "The Darwin Software Engineering Measurement Appliance", Cylant, [www.cylant.com](http://www.cylant.com)
- [21] William R. Dillon, Matthew Goldstein, Multivariate Analysis: Methods and Applications, Wiley-Interscience, August 1984, ISBN: 0471083178
- [22] Computer Associates, "AllFusion Harvest Change Manager Features, Descriptions & Benefits", Feb. 11, 2002, available at: [http://www3.ca.com/Files/FactSheet/af\\_harvest\\_cm\\_fdb.pdf](http://www3.ca.com/Files/FactSheet/af_harvest_cm_fdb.pdf)