Fault-Based Analysis: How History Can Help Improve Performance and Dependability Requirements for High Assurance Systems

Jane Huffman Hayes
Elizabeth Ashlee Holbrook
Inies Raphael C.M.
University of Kentucky Dept. of Computer Science
hayes@cs.uky.edu, {ashlee|irchem2}@uky.edu

David M. Pruett
Geocontrol Systems
Incorporated
david.m.pruett1@jsc.nasa.gov

Abstract

Performance and dependability requirements are key to the development of high assurance systems. Fault-based analysis has proven to be a useful tool for detecting and preventing requirement faults early in the software life cycle. By tailoring a generic fault taxonomy, one is able to better prevent past mistakes and develop requirements specifications with fewer overall faults. Fewer faults within the software specification, with respect to performance and dependability requirements, will result in high assurance systems of improved quality.

1. Introduction

One needs only look to the increasing importance of computer systems in our society and the increasingly trusted roles that they play to understand how important it is to ensure that high assurance systems are correctly built. For example, power generation has largely become digital (instrumentation and control systems for nuclear power plants are becoming increasingly digital) as well as aviation (autopilot and other important systems of large, commercial aircraft are now digital). Clearly we need these software systems to be dependable. Performance also plays an important role in many software systems (many functions are time critical).

In order to ensure that high assurance software systems possess the necessary performance and dependability qualities, it is essential that their requirements are specified correctly. Though performance and dependability are non-functional requirements, they are still requirements and hence can potentially be improved using methods and techniques that have been proven to have general applicability to the requirements domain. In this paper, we argue that a process called fault-based analysis can be applied to performance and dependability requirements for high assurance systems.

The paper is organized as follows. Section 2 presents an overview of fault-based analysis. Related work is discussed in Section 3. Our position is outlined in Section 4 as well as some preliminary results. Finally, conclusions and future work round out the paper in Section 5.

2. Fault-based analysis

"The IEEE standard definition of an error is a mistake made by a developer. An error may lead to one or more faults [13]. To understand Fault-Based Analysis (FBA), a look at a related technique, Fault-Based Testing (FBT), is in order. Fault-based testing generates test data to demonstrate the absence of a set of pre-specified faults. There are numerous FBT techniques. These use a list of potential faults to generate test cases, generally for unit- and integration-level testing [20,3]. Research has been performed in the area of software safety fault identification [20, 6], including research into numerous fault analysis techniques such as Petri-net safety analysis [16,17], Failure Mode, Effects, Criticality Analysis (FMECA) [19], and criticality analysis [29]." [10] Similar to FBT, FBA identifies static techniques, such as traceability analysis, and specific activities within those techniques that should be performed to ensure that a set of pre-specified faults do not exist. Fault-based analysis is risk-driven, and attempts to select V&V techniques to apply in order to best achieve a project's goals. In performing FBA, one targets the strongest fault class or classes [12]. A more extensive survey of related work, such as orthogonal defect classification [4] can be found in [8].

Fault-based analysis, as applied to requirement faults, can help prevent and/or detect faults early in the software lifecycle, resulting in significant cost savings [1]. In earlier work by Hayes [8], a generic fault taxonomy was selected as the basis for requirements FBA, requirements faults were examined, and a method for extending a taxonomy was developed and implemented. Historical data can be used to determine the fault types that are most likely to be introduced and risk analysis can

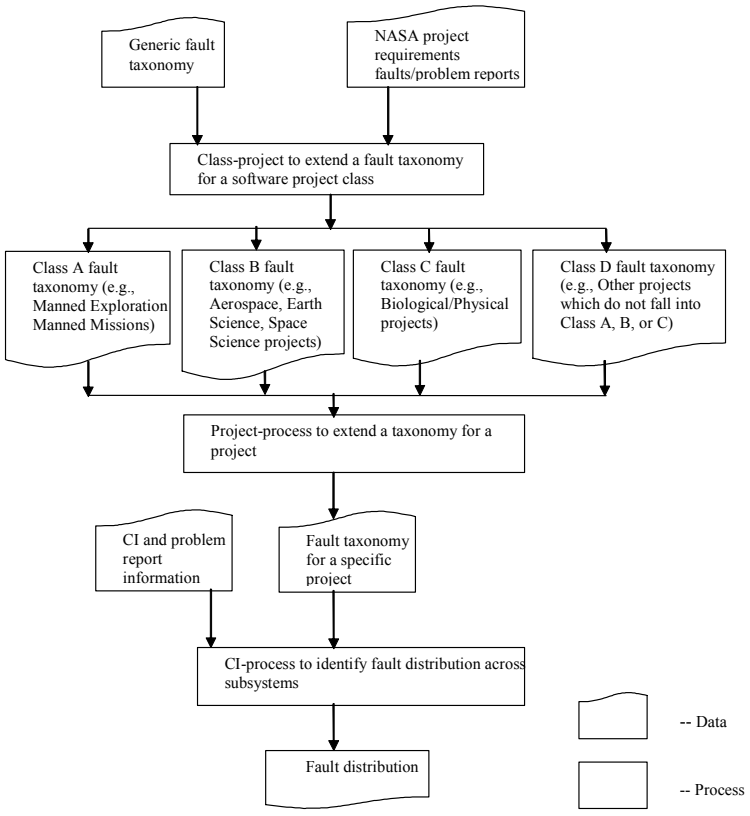

Figure 1: High level process for extending fault taxonomies.

be performed to determine the fault types that would be most devastating if overlooked.

We can identify techniques that help reduce the risk of, prevent, and detect prevalent or targeted fault types. These techniques are then applied as part of the V&V (Verification and Validation) and/or IV&V (Independent Verification and Validation)effort [8]. To provide a requirements-based fault analysis approach, an overall methodology was defined [5]: (i) build a requirement fault taxonomy and a process for tailoring it; (ii) build a taxonomy of V&V techniques and build a matrix of their validated fault detection capabilities; and (iii) develop guidance to V&V agents and software projects for use of the fault-based analysis methodology and assist in its adoption. Just as project quality improves when one prioritizes requirements [7], quality increases when a mechanism is in place to prioritize fault countermeasures and choose the most effective fault detection and prevention methods.

Note that we require defect reports related to requirements. This is the starting point for the FBA process. If starting a new release for an existing system (writing requirements for the new release), one looks at the historical trends of requirements defect reports from the previous releases. If beginning development of a new product, one may look at the historical trends from that organization on earlier products or from related organizations (with personnel overlap, such as management).

3. Tailoring of fault-based analysis

Our position is that FBA can be applied to improve dependability and performance requirements. Improved performance and dependability requirements will result in high assurance systems of improved quality. The US spends approximately $59.5 billion each year on software errors, according to a 2002 Study by the National Institute of Standards (NIST). Lack of software requirements as well as incorrect requirements contributes heavily to this problem [15].

We base our position on prior work with FBA, as applied on various computer software capabilities of the International Space Station [14], a manned flight system and thus high assurance [11].

In developing FBA, we developed a process that can be tailored or applied to: various levels of the software architecture, to historical "slices" of time, etc. Let us examine a few examples of such "tailoring." FBA can be applied at a generic level to collections of related projects, or domains. Given defect information for a collection of projects, the FBA process is applied (as discussed in Section 2) and the result is a tailored taxonomy

Table 1: Configuration item (CI) process for tailoring a taxonomy.

Entry Criteria	Activities	Exit Criteria
1. All inputs are available 2. NASA has authorized use of project data 3. NASA has authorized the taxonomy extension project	1. Select project-specific requirement fault taxonomy 2. Select a CI from the list of project CIs 3. Categorize the fault for the CI according to the project-specific fault taxonomy 4. Determine the frequency of faults for the CI 5. Identify the crucial fault categories for the CI 6. Repeat Steps 2 through 5 for all other CIs	1. A CI-specific requirement fault taxonomy has been developed
Inputs	**Process Controls/Metrics**	**Outputs**
1. Project-specific fault taxonomy 2. Requirement faults/problem reports for the CIs 3. CI-specific information (goals priorities for all CIs	Process Controls: 1. Maintenance of configuration control of taxonomy 2. Maintenance and management of NASA CI data by project Metrics: 1. Person hours for effort 2. Number of CIs 3. Number of faults 4. Historic probability of occurrence 5. Fault exposure values	1. Frequency counts of faults 2. Crucial fault categories for the CI 3. Prioritized fault list for the CI

(fault frequency occurrence) that helps to characterize the domain as a whole. For example, we examined the fault profiles for a number of high assurance NASA systems, applied the FBA process, and created a tailored fault taxonomy for NASA Class A systems (NASA's term for high assurance systems) [11]. Next, we wanted to apply FBA to a specific project. We took the Class A taxonomy, project specific information, and applied our process. The result was a taxonomy tailored to the International Space Station software systems. If historical data is not available, due to the novelty of an application area, one may rely on similar projects or may begin the tailoring process on subsequent maintenance phases of the project. Information extracted from end-users as well as testing reports may be used to benefit fault-based analysis.

4. Position

We are confident that our process can be applied to focus on non-functional requirements such as performance and dependability. To illustrate how our process is applied, one can examine the process used to generate a taxonomy and historical data for the requirements of International Space Station Computer Software Configuration Items (CSCI).

Our process for tailoring a taxonomy is presented in Table 1. Table 2 shows the project categorization percentage data for the International Space Station as a whole. By examining available data and performing trend analysis, we are able to tailor taxonomy to prioritize the particular fault areas that have been historically significant. Table 3 shows our tailored taxonomy for the particular configuration items we have examined [8].

Once we have obtained the historical fault profile, we meet with engineers to determine why the trends have occurred. This begins with a discussion that revisits the timeline of development (possibly for multiple configuration items) and encourages the engineers to think of reasons or causes for the visible trends. One such recent analysis for ISS uncovered several findings related to requirement elicitation between requirements engineers and "specialty engineers." We provide one such example here: "The engineers felt that several of the prominent fault categories could be explained by one phenomenon: the occurrence of incomplete (category 1.1), omitted or missing (1.2), incorrect (1.3), or ambiguous (1.4) requirements is indicative of a lack of engineers, knowledgeable in the thermal, power, environmental, etc. systems, working on these particular requirements" [9].

After discussions with project engineers and examination of previous problem reports, we then developed a common cause tree. A common cause tree is similar to a fault analysis tree, and presents root causes of requirement faults as well as actions

that may be taken to prevent or detect these faults. In the common cause tree, mitigations and corrective actions have been pre-defined to assist a manager in taking measures to improve the requirement specification processes. In the case of ISS, three common causes are of note: noncompliant process, lack of understanding, and human error. Countermeasures were determined for each common cause. For example, faults caused by noncompliant processes may be remedied through formal process certification, effective question and answer processes, more managerial involvement, and trained staff at each certification level.

Table 2: ISS Project categorization percentage data.

Major Fault	% of ISS Faults by Category
.1 Incompleteness	0.209
.2 Omitted/Missing	0.329
.3 Incorrect	0.239
.4 Ambiguous	0.061
.5 Infeasible	0.014
.6 Inconsistent	0.047
.7 Over-specification	0.063
.8 Not Traceable	0.014
.9 [Reserved for future]	---
.10 Non-Verifiable	0.005
.11 Misplaced	0.007
.12 Intentional Deviation	0.007
.13 Redundant/Duplicate	0.005

Table 3: Tailored taxonomy for the ISS CIs.

Major Fault	% of CI Faults by Category
.1 Incompleteness	0.233
.2 Omitted/Missing	0.108
.3 Incorrect	0.301
.4 Ambiguous	0.130
.5 [Reserved for future]	---
.6 Inconsistent	0.130
.7 Over-specification	0.011
.8 Not Traceable	0.023
.9 [Reserved for future]	---
.10 [Reserved for future]	---
.11 Misplaced	0.011
.12 Intentional Deviation	0.023
.13 Redundant/Duplicate	0.023

5. Conclusions and future work

Our approach yields a number of benefits: a) historical information on the types of problems or faults is available; b) this can lead to "lessons learned" that can help improve requirement writing; c) the development of a common cause tree helps identify remedial and proactive actions that can be taken for short and long term improvement; d) historical discussions with engineers give insight into interactions of parallel configuration item developments that may lead to problems; and e) when performing trend analysis on historical data, interaction between development team members and the requirements specification team allows the requirements specification team members to gain further insight into areas where faults have arisen in past projects. By discussing past faults, requirements specification becomes a learning process by which faults may be avoided.

As we have tailored our process to many different dimensions of the software lifecycle, architecture, etc., we are confident that the process can be tailored to functional and non-functional requirements, specifically the non-functional requirements of dependability and performance.

In conclusion, we posit that fault-based analysis approach can help improve the quality of performance and dependability requirements for high assurance systems. We have shown our ability to tailor our approach to a number of different aspects of a software system or project and have thus demonstrated the ease of tailoring of our approach to dependability and performance requirements [11]. There may be some work involved to realize this application. For example, in order to apply FBA to performance and dependability requirements, it will be necessary to be able to identify such requirements. We have had some success in building a classifier to categorize requirement related defect reports according to our taxonomy. We envision using similar techniques to categorize textual requirements that relate to performance or dependability. Additionally, we plan to examine our process to see if it should be modified in order to tailor by requirement type at either a high level (functional and non-functional requirements) or at a lower level (performance, dependability, etc.).

6. Acknowledgments

Our work is funded by NASA under grant NNG04GA38G. Our thanks to Pete Cerna, Kenny Todd, Mike Norris, Bill Gerstenmaier, Bill Panter, Marcus Fisher, and the International Space Station project. We thank Andrea Hunt, Tyler Mueller, and Olga Dekhtyar for their assistance.

7. References

[1] Boehm, B. Software Engineering Economics. Prentice-Hall, Inc., 1981.

[2] Cha, S. S., Leveson, N. G., and Shimeall, T. J. 1988. Safety Verification in Murphy Using Fault Tree Analysis. In Proc. of the 10th International Conference on Software Engineering (Singapore, April 11 - 15, 1988). International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 377-386.

[3] Chen, T. and Lau, M., "Test Suite Reduction and Fault Detecting Effectiveness: An Empirical Evaluation," Lecture Notes in Computer Science, Volume 2043, Springer-Verlag, pp. 253 – 265.

[4] Chillarege, R., Bhandafi, I., Chaar, J., Halliday, M., Moebus, D., Ray, B., and Wong, M. "Orthogonal Defect Classification A Concept for In-Process Measurements," 1EEE TSE, vol. 18, no. 11 (Nov. 1992), pp. 943-956.

[5] Davis, A. Software Requirements: Analysis and Specification. Prentice-Hall, Inc., New York, 1990.

[6] Firesmith, D.G, "Engineering Safety-Related Requirements for Software-Intensive Systems," tutorial H6 at the 27th International Conference on Software Engineering in Saint Louis, Missouri, 15-21 May 2005.

[7] Firesmith, D.G, "Prioritizing Requirements," in Journal of Object Technology, vol. 3, no. 8, September - October 2004, pp. 35-47. http://www.jot.fm/issues/issue_2004_09/column4.

[8] Hayes, J.H. "Building a Requirement Fault Taxonomy: Experiences from a NASA Verification and Validation Research Project," Proceedings of the International Symposium on Software Reliability Engineering, ISSRE 2003, pp. 49 – 59, Denver, CO, November 2003.

[9] Hayes, J.H., Holbrook, A., UKY and Science Applications International Cooperation. "Final Report for Fault-Based Analysis: Improving Independent Verification and Validation (IV & V) through Requirements Risk Reduction." UK-05005. 30 September 2005.

[10] Hayes, J.H, UKY and Science Applications International Cooperation. "Final Report for Fault-Based Analysis: Improving Independent Verification and Validation (IV & V) through Requirements Risk Reduction." UK-04001. 31 October 2004.

[11] Hayes, J.H, Raphael, I.C.M., Pruett, D., Holbrook, A. A Case History of International Space Station Requirement Faults. October 2005. (TR 634-05).

[12] Kuhn, D.R. Fault classes and error detection capability of specification-based testing. ACM Transactions on Software Engineering and Methodology (TOSEM) Volume 8 , Issue 4 (October 1999).

[13] IEEE Standard Glossary of Software Engineering Terminology. IEEE Std. 610.12-1990, 1990.

[14] National Aeronautics and Space Administration International Space Station. http://spaceflight.nasa.gov/station/.

[15] Laplante, P.A. Software Engineering for Image Processing Systems. CRC Press, New York: 2003.

[16] Leveson, N., and Stolzy, J., Safety Analysis Using Petri Nets, IEEE Transactions on Software Engineering, SE-13(3), 1987.

[17] Leszak, M, Perry, D.E., and Stoll, D. A Case Study in Root Cause Defect Analysis, Proceedings of the 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland, 2000, pp.428-437.

[18] Lutz, R. "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems", Proc. RE'93: First 1EEE International Symposium on Requirements Engineering, January 1993, 126-133.

[19] MIL-STD-1629A, Notice 2, Military Standard, Procedures for Performing a Failure Modes Effects and Criticality Analysis, Department of Defense, Washington, D.C., November 28, 1984.

[20] MIL-STD-2167A, Military Standard, Defense System Software Development, Department of Defense, Washington, D.C., February 29, 1988.

[21] Mojdehbakhsh, Ramin, "Software Lifecycle and Analysis Techniques for Safety-Critical Computer-Controlled Systems," Dissertation, George Mason University, 1994.

[22] Morell, Larry, "Theoretical Insights into Fault-based Testing," Proceedings of the Second Workshop on Software Testing, Verification, and Analysis 1998, 19 – 21 July 1998, pp. 45 – 62.

[23] Munson, J.C., Nikora, A.P. Toward a quantifiable definition of software faults. 13th International Symposium on Software Reliability Engineering, 2002, p. 388 –395.

[24] Offutt, J. and Hayes, J. H. "A Semantic Model of Program Faults," In Proc. of the International Symposium on Software Testing and Analysis, pages 195-200, ACM, San Diego, California, January 1996.

[25] Rothermel, G., Harrold, M.J., Analyzing Regression Test Selection Techniques. IEEE Transactions on Software Engineering, 22(8), Aug. 1996.

[26] von Mayrhauser, A., J. Wang, M.C. Ohlsson and C. Wohlin, Deriving a Fault Architecture from Defect History, In Proc. of the International Symposium on Software Reliability Engineering, ISSRE99, pp. 295-303, November 1999, Boca Raton, Florida, USA.

[27] Wallace, D., and Fujii, R., Software Verification and Validation: An Overview, IEEE Software, Volume 6, No. 3, May 1989.