

# Patterns of Software Defect Data on Spacecraft

Robyn R. Lutz\*  
Jet Propulsion Laboratory  
and Iowa State University  
[rlutz@cs.iastate.edu](mailto:rlutz@cs.iastate.edu)

Inés Carmen Mikulski  
Jet Propulsion Laboratory  
Pasadena, CA 91109-8099  
[ines.c.mikulski@jpl.nasa.gov](mailto:ines.c.mikulski@jpl.nasa.gov)

## Abstract

*The identification of patterns of software defect data yields insights into improving the quality of both operational and future spacecraft. Recent investigations of software defect data at Jet Propulsion Laboratory has revealed both expected and unexpected patterns of defect data. This paper describes the results of applying this technique to both post-launch (operational) and pre-launch (developmental) spacecraft. It then describes four key challenges that remain to achieving fuller utilization of defect analysis in future systems.*

## 1. Introduction

The identification of patterns of software defect data yields insights into improving the quality of both operational and future spacecraft. This report describes (1) the technique we developed to mine patterns from JPL problem reporting databases and (2) the results of applying this technique to both post-launch (operational) and pre-launch (developmental) spacecraft.

The defect analysis technique used is an adaptation to spacecraft of an approach originally developed at IBM. The technique, called Orthogonal Defect Classification (ODC), has been widely used in industry to identify patterns in defect databases without incurring significant additional costs [1].

This paper reports results from two applications of the adapted ODC technique at JPL. In the first application, nearly two hundred post-launch problem reports of critical software ISAs (Incident/Surprise/Anomalies) on seven spacecraft were analyzed. Since the goal of the research is to provide a sound, quantitative foundation to enable improvements, a formalized pilot study approach (the rigorous Glass criteria) was used. Among the unexpected patterns reported here are: (1) outdated or missing procedures were involved in one-fifth of these critical incidents and (2) requirements changes were rarely due to previous requirements having been incorrect. Instead, the post-launch changes involved new requirements for the software to handle rare events or to compensate for hardware failures or limitations. The ODC analysis of the critical post-launch software anomalies on these spacecraft generated a set of process recommendations that are described in the paper.

The second application of the adapted ODC technique at JPL was the analysis to date, in collaboration with the Mars Exploration Rovers (MER) project, of some three hundred problem reports generated during testing. Among the useful findings was that "false positive" problem reports (where testing personnel believe that the software is behaving incorrectly when it is, in fact, behaving correctly) provide a degree of "crystal ball" forecasting regarding which features of the software are likely to confuse operational personnel. This knowledge can then be applied via training or targeted documentation to forestall similar problems post-launch.

This paper describes unexpected patterns such as this, as well as patterns of defect data that confirm existing testing and operational assumptions on the spacecraft. The results appear to be broadly applicable to reducing software defects that propagate to post-launch in future space missions.

---

\*The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. It was funded by NASA's Office of Safety and Mission Assurance, Center Initiative UPN 323-08. The first author's research is supported in part by National Science Foundation Grants CCR-0204139 and CCR-0205588.

The rest of the paper is organized as follows. Section 2 describes the approach used to analyze the defects and find the patterns. Section 3 presents and evaluates the results, and discusses some consequences of these results for the testing and operations of high-consequence systems. Section 4 briefly describes related work in defect analysis and the requirements engineering of safety-critical systems. Section 5 describes four key challenges to be faced in future work. Section 6 concludes the paper.

## 2. Approach

Figure 1 shows an overview of the approach that we have taken in our study of defect data.

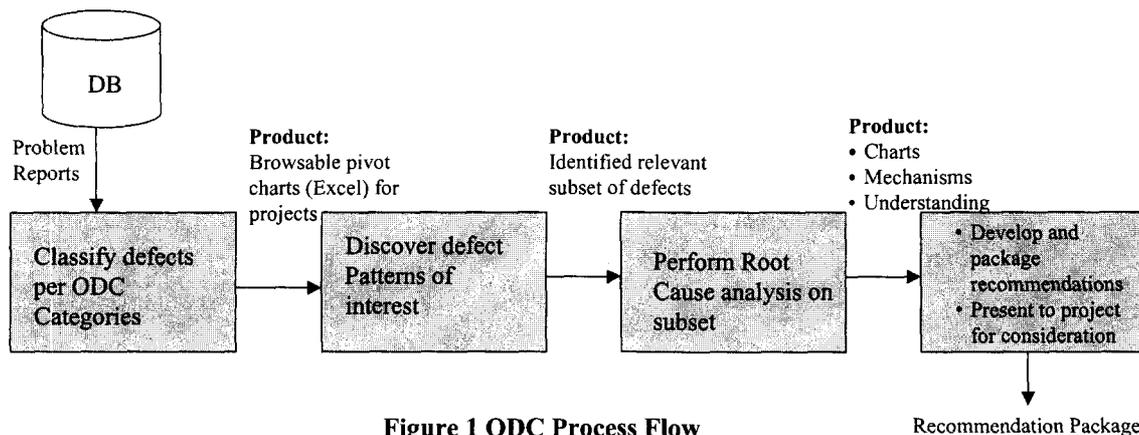


Figure 1 ODC Process Flow

The data source used was software problem reports from a multi-mission, institutional database. The work reported here consists of two studies to date. The first study analyzed the post-launch software problem reports from seven spacecraft. These Incident/Surprise/Anomaly (ISA) reports are referred to here by the shorthand term “anomaly reports.” They document the occurrence of unexpected behavior as well as near-misses and actual defects during operations.

The anomaly reports document not only defects but also any behavior that is unexpected by the testing or operational personnel. The anomaly reports are thus a rich source of discovery of both latent requirements (where the software does not behave correctly in some situation) and of requirements confusion (where the software’s behavior was correct but confused the tester or operator).

The focus of this first study was on the post-launch anomaly reports classified by their projects as critical (e.g., “red-flag,” “potential-red-flag,” “high-criticality,” etc.). There were 199 such problem reports on the seven spacecraft studied. The spacecraft were the Galileo mission to Jupiter, launched in 1989; Mars

Global Surveyor, launched in 1996; Cassini/Huygens, launched in 1997 to Saturn and Titan; Deep Space 1, an ion-propulsion and remote-agent technology mission, launched in 1998; Mars Climate Orbiter, launched in 1998; Mars Polar Lander, launched in 1999; and Stardust, a comet sample-return mission, also launched in 1999.

The second study extended the previous work backward in time to analyze the problem reports generated during integration and system testing on a current spacecraft, the twin Mars Exploration Rover spacecraft (MER), to be launched in May and June, 2003. These problem reports are called Problem/Failure Reports or Developmental Problem/Failure Reports. They are referred to here by

the shorthand term “problem reports.” To date, 311 such problem reports have been analyzed in this way. (This is roughly the Avionic Integration and Test reports, and the Assembly, Test, Launch and Operations reports, generated between April, 2002 and Jan, 2003, that currently contain information in the corrective action field).

The approach taken to the work reported here was to analyze the anomaly reports generated during testing and operations to gain insight into patterns of defects. The high-level goal of this work is to reduce the number of safety-critical software anomalies that occur after launch.

Although the data format available in the problem-reporting database differed somewhat among projects, all the problem reports contain the following three parts: a description of the problem, a subsequent analysis of the problem, and a description of the corrective action taken to close out the anomaly report.

The method of analysis for the anomaly reports was an adaptation of Orthogonal Defect Classification (ODC) [1]. ODC is a defect-analysis technique developed at IBM in the 1980’s and widely used in

industry to measure both developmental and operational defects. ODC provides a way to “extract signatures from defects” and to correlate the defects to attributes of the development process. We describe the approach briefly here and refer the reader to [11] for a fuller description of the classification method and steps taken to avoid bias in the results.

The ODC-based approach uses four attributes to characterize each defect:

- Activity, which describes when the defect occurred (e.g., System Test or Flight Operations)
- Trigger, which describes the environment or condition that had to exist for the defect to surface (e.g., a Fault Recovery condition or Hardware/Software interaction)
- Target, which characterizes the high-level entity that was fixed in response to the defect’s occurrence (e.g., Flight Software or Information Development), and
- Type, which describes at a lower-level the actual fix that was made (e.g., Documentation, Procedure, or Function/Algorithm).

For the post-launch anomalies, “Criticality” was a fifth, implicit attribute (since only critical anomalies were studied). For the testing anomalies, “Release number” was added as an attribute.

Two analysts classified each anomaly to reduce bias. Any disagreements regarding classifications were reconciled during regular joint discussions. Spacecraft personnel, especially on MER, generously assisted us by answering domain and process questions. The results of the ODC analysis were entered manually into an Excel file. Pivot table charts were then produced in Excel. These charts summarized and helped visualization of the results in various cross-correlations (e.g., Activity x Target x Type, Trigger x Target). The

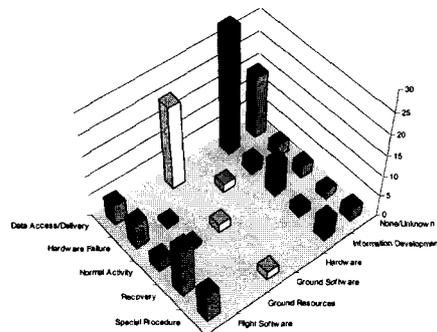


Figure 2

data among spacecraft or phases could also be automatically normalized for the displays if desired. Figures 2 and 3 show examples of the charts thus produced.

The second step in the analysis process (see Fig. 1) is identifying defect patterns of interest. These patterns are discovered in three ways. (1) The project asks questions of interest, such as “how do the number of PFRs of type “assignment/initialization” (i.e., fixed by changing the software in this way) compare among releases in ATLO?” (2) The analyst or project speculates about an expected pattern of data and searches the pivot tables to confirm/reject this speculation. For example, we expected that most critical operational anomalies occurred during critical mission phases, but found that this assumption was false. (3) The analyst browses the pivot table charts for surprising peaks or valleys. For example, there were many more problem reports with ODC Type = “Nothing Fixed” (i.e., no corrective action) than we had expected.

When a pattern of interest in the defects is discovered, the question is often “why? That is, the reason for the pattern is sometimes not apparent. The third step (Fig. 1) is then to perform a more detailed analysis of each of the defects in the specific subset of interest in order to explain the pattern. This step is very similar to root cause analysis [7]. An advantage of ODC is that it minimizes root cause analysis, which tends to be time-consuming and costly. With the ODC approach, root cause analysis only needs to be performed for the subset of the defects identified as belonging to an unexplained pattern of concern. For example, to understand why so many problem reports had no corrective action taken, we performed root-cause analysis on the subset of problem reports described in the previous paragraph. The results are described in Section 3.1.

Once an understanding of the underlying causes of

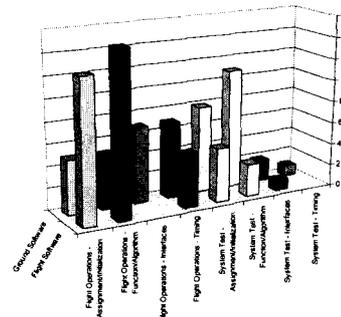


Figure 3

the pattern of defects emerges, the fourth step of the process is to translate this understanding into project or process recommendations to avert this undesirable pattern in the future.

For example, the ODC analysis of the post-launch critical software anomalies showed the unexpected pattern that many anomalies were being fixed by changes to procedures. Root cause analysis showed that this was because some needed procedures did not exist. The recommendation was that a checklist be maintained of operational procedures needed on previous, similar spacecraft, and that comparison with this list be performed as part of the operational readiness for any new spacecraft.

### 3. Results and analysis

We describe in this section some defect patterns that emerged from the ODC analysis, both during testing and during operations, and describe the consequences of our findings. Initial testing results have been previously reported in [11]; earlier post-launch results have been reported in [9] and [10].

#### 3.1 Testing

Some patterns that we identified in the documented testing problem reports were expected. For example, more defect reports were generated in integration testing than later on in system testing, a typical pattern. Similarly, the percentage of defects triggered by the testing of a single command, as opposed to the behavior induced by testing a sequence of commands, was greater in integration testing than in system testing, another typical pattern in testing.

Other patterns of defect data from testing were unexpected. We describe two of them here with the caveat that this work is on-going (i.e., problem reports are still being generated and analyzed).

One unexpected pattern was that 70 of the problem reports had ODC Target = "None/Unknown" and Type = "Nothing Fixed." That is, about one-quarter of these problem reports resulted in no corrective action. The reason for this unexpected result can be further investigated with root cause analysis to understand the reason for the pattern. In this case, the pattern often seemed to be due to bugs that were no longer present in the software releases currently being tested by the time that the problem report was investigated. Lending credence to this analysis is the fact that of the 70 such problem reports, only eight occurred in the later system

testing and the other 62 occurred earlier in integration testing. There is some risk, of course, in the assumption that if the defect no longer occurs in the current release under the same circumstances, that no corrective action is necessary. Often, however, the software and system have evolved to the point where the conditions under which the defect originally occurred cannot be duplicated.

The root cause analysis also showed that some of the problem reports in this pattern were not fixed for another reason—that the software was, in fact, functioning correctly but that the tester believed that the software was behaving incorrectly. The implications of these problem reports for operations is discussed below.

A second unexpected pattern of defect data was that 80 of the 274 problem reports fixed by changes to flight software had an ODC Type = "Function/Algorithm." A hypothesis to explain why this pattern occurred is that it reflects the evolving (i.e., unstable) requirements, especially with respect to latent software requirements emerging from changes to, or interfaces with, hardware components or other software components. Root cause analysis to confirm or reject this hypothesis awaits completion of additional problem reports.

#### 3.2 Operations

Table 1 summarizes four unexpected patterns encountered in the critical software anomaly reports of the seven launched spacecraft. For example, changes or updates to procedures were involved in the fix to many critical anomalies (23%). This suggests that, for future spacecraft, assurance that needed procedures are in place may be high-yield in terms of reducing critical anomalies.

Similarly, we did not expect that ground software would be involved in the corrective action of so many critical anomalies (22%). This result indicates that verification of ground software's correctness is essential for preventing critical anomalies.

Another finding of interest was that 41% of the critical post-launch software anomalies were triggered by problems with data access or delivery (e.g., uplink or downlink difficulties). It is well known that problems occur in the interfaces between the systems, but technical difficulties with downlink caused a surprising number of critical anomalies.

<b>Examples of Unexpected ISA patterns:</b>	<b>Process Recommendation:</b>	<b>Example (from spacecraft):</b>
23% of critical ISAs had <u>ground software</u> as Target (fix)	Software QA for ground software	Unable to process multiple submissions. Fixed code.
30% of critical ISAs had <u>procedures</u> as Type	Assemble checklist of needed procedures for future projects	Not in inertial mode during star calibration. Additions made to checklist to prevent in future.
Of these, 35% had <u>Data access / delivery</u> as Trigger	Better communication of changes and updates to operations	Multiple queries for spacecraft engineering and monitor data failed. Streamlined notification to operators of problems.
34% of critical ISAs involving system test had software configuration as Trigger (cause) ; 24% had hardware configuration as Trigger	Additional end-to-end configuration testing	OPS personnel did not have a green command system for the uplink of two trajectory-correction command files. Problems resulted from a firewall configuration change.

**Table 1 Summary of Unexpected Patterns**

We found, as well, that the occurrence of “rare” events that merited additional fault protection requirements triggered about one-third of the anomalies. This suggests that effort spent during development on analyzing failure scenarios is effort well-spent, since rare events do, in fact, cause critical anomalies.

In addition, the anomaly reports showed that many new requirements post-launch involved software compensation for degraded hardware functionality. What broke (hardware) sometimes wasn’t what got fixed (software). This seems to indicate that pre-launch contingency planning for post-launch scenarios in which some hardware functionality can be passed to software may help lower risk.

#### 4. Related Work

The work described here draws on related work in several areas. We describe those areas briefly here and refer to the reader to [11] for fuller accounts of recent work in these areas.

In the area of defect analysis, our work is based on the ODC technique introduced by Chillarege et al. [1]. The ODC technique has been used primarily for defect analysis of systems in development and testing, but also for defect-analysis of deployed systems [2], as we do with the operational spacecraft. The work described here extends previous work on the ODC technique to the spacecraft domain.

Root-cause analysis is another defect analysis technique that has been widely used. Its disadvantage is that it is labor-intensive (19 minutes per defect in the

recent study by Leszak, Perry, and Stoll [7]), so it is usually applied only to a subset of defects and is difficult to automate. This contrasts with the method described here, which is currently performed manually at 4-8 minutes/defect and supports partial automation (see section 5.1).

Our results confirm the findings of some previous researchers that requirements-related defects predominate in critical systems [4,8]. A recent study by Lauesen and Vinter has also found similar results for non-critical systems, with slightly more than half the defect reports being requirements defects [6]. Our results also confirm recent work that implicates requirements misunderstanding in accidents [12].

#### 5. Challenges

The results described above show that detection of patterns in defect data is both feasible and useful. As we look toward future work, four key challenges remain. We here describe each of these challenges and suggest approaches to addressing them.

##### 5.1 Automation

Projects need near-real-time defect analysis for trend analysis and planning, both in testing and in operations. Without automation, such timely defect analysis is impossible.

Partial automation is quite feasible and, in fact, can be supported by existing or planned problem-reporting systems. These systems require the initiator of the defect report, as well as the analyst and the implementer of the corrective action, to classify the

defect and the fix from among a list of possible categories. Standardization and compliance within a project can provide project-level data; standardization and compliance within an organization can provide organization-wide data. One of our goals in this effort is to make clear to the projects and the organizations the benefits that can be reaped by such partial automation.

Full automation is, however, a different issue. Our work has shown clearly that some tradeoffs exist between automation and accuracy. This is true for two reasons. First, defects are often not simple. However, defect-classification techniques, of necessity, impose a simplified structure on the defect report. A defect may have multiple triggers or causes, as well as multiple targets or corrective actions. The classification, on the other hand, usually requires the selection of one trigger and one target, or at least requires a choice as to which trigger and target are most important. If these selections are done at the time that the defect occurs, they may be done on the basis of partial, sometimes erroneous, information.

Second, defect reports are rich sources of information regarding near-misses, speculations by domain experts as to possibly related occurrences, and confusions regarding programmed vs. expected behavior. Automation tends to “flatten” the defect report, removing these text-based insights. These two reasons lead us to recommend that full automation (i.e., removal of textual descriptions) not be sought. Partial automation to detect patterns of interest is recommended, but the fuller investigation into why these unexpected patterns occur (i.e., the root cause analysis of subsets of interest) requires the richness of information only available in the text-based description.

## 5.2 Product-Line Perspective

Many defect reports contain “forward-pointers” to the anticipated recurrence of the same or similar problem on future systems. For example, a defect report may note that the same interface defect found on one system may be able to recur in other systems using the same hardware unit. Both those testing and operational personnel who diagnose the reported defects, and those who determine and implement the corrective actions, regularly call out the possibility that the same problems will recur in similar systems. Essentially, these individuals, who are experts in this domain, demonstrate a product-line perspective.

This broader focus is evident even when no fix is recommended on the current system (e.g., because the scenario of concern will never occur again on this

particular mission). Even in these cases, attention is often drawn to the possibility of a similar problematic configuration occurring on other systems in the product line.

Although such information is captured in the problem reports, it may never be retrieved and used. Better means (i.e., less ad-hoc techniques) are needed to mine the problem-reporting database for other, future systems in the same product line. As product-line practices continue to grow, the availability and tracking of product-line defect insights can reduce risk in similar systems.

## 5.3 Integration with run-time monitoring

Traditionally, run-time monitoring during operations has been considered to be the first line of defense against defects, with the occurrence of a defect indicating the failure or limitations of run-time monitoring. Pairing of run-time monitoring with operational defect analysis may be able to enhance the capabilities of run-time monitoring by identification of defect patterns of concern.

Defect patterns may be able to provide updated input to run-time monitors, perhaps driving updates to the monitors’ parameters. The timeliness of the defect analysis is, of course, a key question and requires the kind of automation discussed above.

Interest in better integration of run-time monitoring with defect analysis is especially important because systems on extended missions and highly autonomous systems will depend ever more heavily on run-time monitors to achieve robustness.

## 5.4 Testing as a simulation of operations

Our analysis found that misunderstandings by testers regarding what the behavior of the software should be (i.e., the requirements) are common. Sometimes the resulting problem reports lead to improved documentation of requirements, of design-decision rationales, or of procedures. Often, however, the reports are closed with no action taken. These problem reports are “false positives” with the software behaving correctly but the testing personnel not recognizing that fact.

We found that such misunderstandings can recur fairly regularly. These confusions consume time and resources (e.g., labor, test facilities). Mismatches between the system behavior expected by operational personnel and the actual system behavior (even if correct) adds risk to the mission. Weiss et al., for

example, after studying several recent aerospace accidents, stated, “software-related accidents almost always are due to misunderstanding about what the software should do” [12].

We have recommended that those “false-positive” problem reports found in testing that meet the following criteria not be closed without additional documentation or training: (the scenario or configuration could recur in operations) AND (the misunderstanding could recur in operations) [11].

An example of such an anomaly was when two different representations of the same time, although correctly used in the software, caused a reasonable misunderstanding during testing that could recur during operations. Since the testing personnel’s misunderstanding could readily be repeated by operational personnel or maintenance programmers during the mission, the problem report merits additional documentation before closure. Long-lived missions with turnover of personnel and consequent loss of knowledge may be particularly vulnerable to the risk of recurring requirements misunderstanding in operations.

In addition, certain classes of requirements misunderstandings recurred in the problem reports. For example, there were several instances where subtle differences between, e.g., unresponsive and unavailable component states, or between the duration of a measured value and the duration of its current high-water mark, confused testing or operational personnel. It may be feasible to target specific classes of requirements confusions to receive additional verification, much as the verification of requirements for flight control systems currently involves awareness of common mode confusions by pilots. Work is ongoing in this area.

## 6. Conclusion

Analysis of software defect data from testing and operations on these eight spacecraft has identified some surprising patterns. These results point the way to the next set of challenges to be addressed by future defect analysis. The challenges described in Section 5 suggest that new models of defect analysis may be needed—to automate defect analysis without losing the domain insights contained in text-based reporting, to incorporate a product-line perspective, to integrate operational defect reporting with run-time monitoring, and to track and remedy requirements confusions in testing before they can recur in operations. Our future work in this area will be guided by these challenges.

**Acknowledgments.** The authors thank Daniel

Erickson and the Mars Exploration Rover engineers and test teams for their assistance and feedback.

## References

- [1] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, “Orthogonal Defect Classification—A Concept for In-Process Measurements,” *IEEE Trans on SW Eng*, Nov. 1992, pp. 943-956.
- [2] S. Dalal, M. Hamada, P. Matthews, and G. Patton, “Using Defect Patterns to Uncover Opportunities for Improvement,” *Proc. Int’l Conf Applications of Software Measurement*, 1999.
- [3] R. L. Glass, “Pilot Studies: What, Why, and How?,” *The Journal of Systems and Software*, vol. 36, pp. 85-97.
- [4] K. S. Hanks, J. C. Knight, and E. A. Strunk, “Erroneous Requirements: A Linguistic Basis for Their Occurrence and an Approach to Their Reduction,” *Proc. 26<sup>th</sup> NASA Goddard Software Eng Workshop*, IEEE, Greenbelt, MD, Nov., 2001.
- [5] S. D. P. Harker, K. D. Eason, and J. E. Dobson, “The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering,” *Proc. IEEE Intl Symp on Requirements Eng*, IEEE Computer Society, Los Alamitos, CA, 1992, pp. 266-272.
- [6] S. Lauesen and O. Vinter, “Preventing Requirements Defects: An Experiment in Process Improvement,” *Requirements Engineering Journal*, 2001, pp. 37-50.
- [7] M. Leszak, D.E. Perry and D. Stoll, “Classification and Evaluation of Defects in a Project Retrospective,” *The Journal of Systems and Software*, vol. 61, issue 3, 1 April, 2002, pp. 173-187.
- [8] R. Lutz, “Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems,” *Proc IEEE Intl Symp Req Eng*, IEEE CS Press, 1993, pp. 126-133.
- [9] R. Lutz and I. C. Mikulski, “Empirical Analysis of Safety-Critical Anomalies during Operations,” submitted, 2002.
- [10] R. Lutz and I. C. Mikulski, “Operational Anomalies as a Cause of Safety-Critical Requirements Evolution,” *The Journal of Systems and Software*, to appear.
- [11] R. Lutz and I. C. Mikulski, “Requirements Discovery during the Testing of Safety-Critical Software,” *Proc of Int’l Conf on Software Eng*, 2003, to appear..
- [12] K. A. Weiss, N. Leveson, K. Lundqvist, N. Farid, and M. Stringfellow, “An Analysis of Causation in Aerospace Accidents,” *Space*, 2001, Aug., 2001.