A Case History of International Space Station Requirement Faults

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

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

Abstract

There is never enough time or money to perform Verification and Validation (V&V) or Independent Verification and Validation (IV&V) on all aspects of a software development project, particularily for complex computer systems. We have only high-level knowledge of how the potential existence of specific requirements faults increases project risks, and of how specific V&V techniques (requirements tracing, code analysis, etc.) contribute to improved software reliability and reduced risk. An approach to this problem, fault-based analysis, is proposed and a case history of the National Aeronautics and Space Administration's (NASA) International Space Station (ISS) project is presented to illustrate its use. Specifically, a tailored requirement fault taxonomy was used to perform trend analysis of the historical profiles of three ISS computer software configuration items as well as to build a prototype common cause tree. ISS engineers evaluated the results and extracted lessons learned.

1. Introduction

In order to reduce the risk associated with software and to verify whether software meets its requirements, Verification and Validation (V&V) activities are performed as a part of the overall development process. The V&V techniques are sometimes performed by a third party, referred to as an Independent Verification and Validation (IV&V) agent. We are never able to perform all the V&V or IV&V activities that we would like to perform. We need a way to ensure that the activities that we do perform will be the most effective at reducing risk for the project. Fault-Based Analysis (FBA) is one way to approach these challenges [5].

The IEEE standard definition of an error is a mistake made by a developer. An error may lead to one or more faults [9]. To understand 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 [17, 2]. Research has been performed in the area of software safety fault identification [16], including research into numerous fault analysis techniques such as petri-net safety analysis [10], Failure Mode, Effects, Criticality Analysis (FMECA) [14], and criticality analysis [24]. Similar to FBT, Fault-Based Analysis identifies static techniques (such as traceability analysis) and even specific activities within those techniques (e.g., perform back-tracing to identify unintended functions) that should be performed to ensure that a set of pre-specified faults do not exist. As part of Fault-Based Analysis, a tailored taxonomy can be developed. Historical data can be used to determine the fault types that are most likely to be introduced or risk analysis can be performed to determine the fault types that would be most devastating if overlooked. Static techniques that prevent or detect these fault types are then applied as part of the V&V and/or IV&V effort [5].

As mentioned above, Fault-Based Analysis has similarities to Fault-Based Testing, where one targets the strongest fault class when designing test generation algorithms in order to increase the effectiveness of the tests without unduly introducing overlap [8]. FBA is also risk-driven, and attempts to select V&V techniques to apply in order to best achieve a project's goals. In this way, it is similar to test case selection during regression testing, where one attempts to reduce the time required to re-test a modified program by selecting a subset of the existing test suite [22]. A more extensive survey of related work, such as orthogonal defect classification [3] can be found in [5].

Our current work focuses on requirements Fault-Based Analysis. A requirement fault is a fault that originates during the requirements phase (e.g., omitted requirement, incomplete requirement description). This paper concentrates on the historical analysis aspect of requirements Fault-Based Analysis. Specifically, we applied our taxonomy tailoring processes from [5] to the requirements faults of three computer software configuration items (CIs) of the International Space

Table 1. ISS requirement fault taxonomy.

Major Fault	Sub-Faults
1. Requirements	Originate in Requirements phase; found in Requirements Specifications
.1 Incompleteness	.1.1 Incomplete Decomposition .1.2 Incomplete Requirement Description
.2 Omitted/Missing	.2.1 Omitted Requirement .2.2 Missing External Constants .2.3 Missing Description of Initial System State
.3 Incorrect	.3.1 Incorrect External Constants .3.2 Incorrect Input or Output Descriptions .3.3 Incorrect Description of Initial System State .3.4 Incorrect Assignment of Resources
.4 Ambiguous	.4.1 Improper Translation .4.2 Lack of Clarity
.5 Infeasible	. ---------------------
.6 Inconsistent	.6.1 External Conflicts .6.2 Internal Conflicts
.7 Over-specification	---------------------
.8 Not Traceable	.----------------------
.9 [reserved for the future]	----------------------
.10 Non-Verifiable	---------------------
.11 Misplaced	----------------------
.12 Intentional Deviation	----------------------
.13 Redundant	----------------------

Station. A highly complex system with available historical data, ISS provides an ideal environment to apply Fault-Based Analysis work to software and system development. We performed trend analysis on the results, then built a small common cause tree and extracted lessons learned.

The paper is organized as follows. Fault-Based Analysis, the requirement fault taxonomy, and the processes for tailoring fault taxonomies and for building the common cause tree are presented in Section 2. The International Space Station case history is discussed in Section 3. Sampling, trend analysis, and lessons learned for the ISS case history are found in Section 4. Finally, Sections 5 and 6 presents lessons learned and directions for future work.

2. Fault-Based Analysis

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 [5], a generic fault taxonomy was selected as the basis for requirements Fault-Based Analysis, requirements faults were examined, and a method for extending a taxonomy was developed and implemented. To provide a requirements-based fault analysis approach, an overall methodology was defined [4]: (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. Here, we focus on tailoring a taxonomy for a given CI and for a given time period (item (i) above) as well as extracting related guidance for the project (item (iii) above). In this section we will present our requirement fault taxonomy, the configuration item process for tailoring a taxonomy, and our preliminary work on common cause analysis.

2.1 Requirement Fault Taxonomy

The Nuclear Regulatory Commission (NRC) requirement fault taxonomy from NUREG/CR-6316 [12] was the basis for our earlier work [5]. We found many papers that confirmed these requirements fault types and found only a few papers that described “new” requirement faults (see [6]). In [5], we examined requirement faults for several NASA software systems. The examination resulted in a number of changes to the taxonomy. The resulting “generic NASA requirement fault taxonomy,” found in Table 1, has thirteen fault categories [5]. Their definitions can be found in [5, 6].

2.2 Configuration Item (CI) Process

The process to determine the fault distribution for a CI is shown in Table 2. The table consists of entry criteria, activities, exit criteria, inputs, outputs, and process controls/metrics. All inputs such as requirement faults and/or problem reports must be available before the process starts. Entry criteria must be met.

Next, activities are performed including selecting a project-specific requirement fault taxonomy, choosing a CI from the list of CIs, examining problem reports or requirement faults of the chosen CI, etc. Note that the CI-process uses the same set of activities as the in project-process, discussed in [5, 6]. Although the activities are the same, they are applied to the specific CI of interest, not to the entire project. The outputs of this process are the fault frequency counts of the faults and the crucial requirement fault categories (those most frequently occurring) for the CI.

We calculate fault exposure values as the product of the tolerance factor and the probability of its occurrence [6]. From the fault exposure values, we create a prioritized list of faults that could have critical effect on the system. We repeat this process for each CI of interest. If certain fault types are found more frequently for a given CI, then it is important to seek improvement

Table 2. 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

in that area and to attempt to prevent and/or detect these fault types [5].

The process controls ensure that all versions of our requirement fault taxonomy are properly maintained under configuration control. Process metrics include person hours for the effort, number of CIs, number of requirement faults, historical probability of occurrence, and fault exposure values [6].

2.3 Common Cause

To determine the feasibility of identifying root causes, a small, prototype common cause tree was constructed. 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 these faults. We examined information found in 25 problem reports (PRs). Problem reports were divided by fault category. Two senior analysts examined the PRs and looked for common causes. They found three: noncompliant process, lack of understanding, and human error. Next, countermeasures were determined for each common cause.

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. Faults with human error as a cause may be avoided by strengthening the question and answer process and by furthering the technical expertise of those on the project. Finally, a lack of understanding of requirements may be countered with a more in-depth technical review process and increased technical expertise. Specifying requirements using clear and precise language, and paying close attention to subtleties of natural language may help avoid misunderstanding and human error [4]. The resulting common cause tree is shown in Figure 1. Further information may be found in [6].

The ISS engineers found the common cause tree to be useful and have requested that we build a much larger tree, providing more detailed causes. We plan to adopt some of the causes delineated by Leszack, Perry, and Stoll [10] in our future efforts.

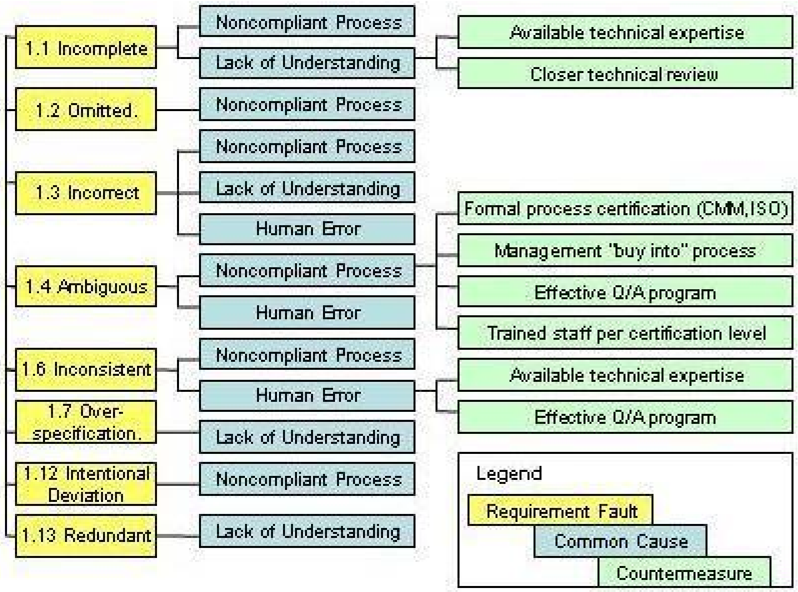

Figure 1. Common cause tree.

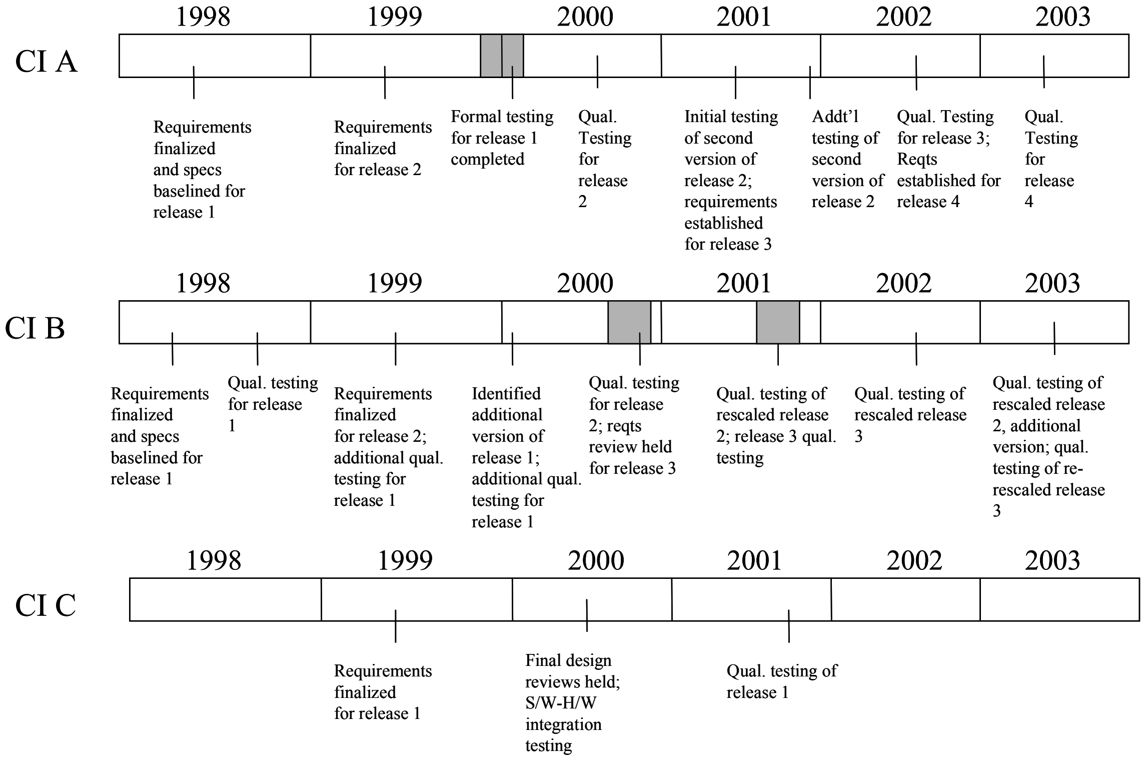
CI A
1998
1999
2000
2001
2002
2003
Requirements finalized and specs baselined for release 1
Requirements finalized for release 2
Formal testing for release 1 completed
Qual. Testing for release 2
Initial testing of second version of release 2; requirements established for release 3
Addt'l testing of second version of release 2
Qual. Testing for release 3; Reqts established for release 4
Qual. Testing for release 4
CI B
1998
1999
2000
2001
2002
2003
Requirements finalized and specs baselined for release 1
Qual. testing for release 1
Requirements finalized for release 2; additional qual. testing for release 1
Identified additional version of release 1; additional qual. testing for release 1
Qual. testing for release 2; reqts review held for release 3
Qual. testing of rescaled release 2; release 3 qual. testing
Qual. testing of rescaled release 3
Qual. testing of rescaled release 2, additional version; qual. testing of re-rescaled release 3
CI C
1998
1999
2000
2001
2002
2003
Requirements finalized for release 1
Final design reviews held; S/W-H/W integration testing
Qual. testing of release 1

Figure 2. Timeline of activities for the Configuration Items of the ISS, 1998 – 2004.

3. International Space Station Case History

The International Space Station (ISS) represents a global partnership of sixteen nations and will have over two million lines of on-board and over ten million lines of ground support software [20]. In our case history, we examined only a subset of this expansive system. The tailored requirement fault taxonomy (project-specific) for ISS is shown in Table 1.

The configuration items (CIs) that we examined are all mission critical portions of the ISS and have real-time capabilities. To maintain anonymity, we will refer to them as CI A, B, and C. CI A is roughly 125,000 lines of code, CI B is roughly 45,000 lines of code, and CI C is roughly 89,000 lines of code. All of the code is written in Ada except for minor efficiency enhancements in low-level code. Note that these CIs were specified, designed, and developed by a large U.S. aerospace contractor who was following Department of Defense standard 2167A [15] software development processes. Requirements are tagged at the "shall" level. The requirements are typically one sentence in length, but often reach multiple levels of indentation showing logical relationships. We examined requirement fault reports from 1998 to 2004.

CI A consists of 430 requirements. All requirements are in the software requirement specification (SRS), referencing paragraphs in the interface control document (ICD) as appropriate. The references are the "shalls" so the actual requirements count is not transferred from the ICDs. The SRS covers releases 1-4 of the software.

CI B consists of 339 requirements for release 1 of the software. The second release of the software implemented 850 requirements, and the current release has about 875 requirements. CI C consists of 339 requirements for release 1 of the software.

A brief history of these CIs, for the problem report time period examined, is depicted in Figure 2. Note that 2004 is not shown or discussed as only partial data was available for that year at the time of the study (only one requirement fault was categorized for that year). In 1998, requirements were being finalized for the initial releases of CIs A and B. A system specification review (SSR) was held in 1998 and the SRS was baselined. The software interface control documents took longer, and were baselined later in 1998.

In 1999, the requirements were finalized for release 2 of CIs A and B and for release 1 of CI C. Formal testing

Table 3. Summary of all CIs by year.

Major Fault	1998	1999	2000	2001	2002	2003	2004	Total	Percentage of ISS Faults by Category
.1 Incompleteness	1 [1,0,0]	16 [2,13,1]	4 [0,4,0]	6 [3,2,1]	9 [1,2,6]	5 [2,2,1]	0 [0,0,0]	41 [9,23,9]	23.30% [12.68%, 30.26%,31.03%]
.2 Omitted/Missing	0 [0,0,0]	7 [3,4,0]	3 [1,2,0]	3 [1,1,1]	5 [2,0,3]	1 [0,1,0]	0 [0,0,0]	19 [7,8,4]	10.80% [9.86%,10.53%,13.79%]
.3 Incorrect	3 [3,0,0]	23 [7,16,0]	7 [3,4,0]	9 [8,0,1]	6 [0,1,5]	4 [0,3,1]	1 [1,0,0]	53 [22,24,7]	30.11% [30.99%,31.58%,24.14%
.4 Ambiguous	0 [0,0,0]	13 [9,4,0]	4 [1,3,0]	2 [1,1,0]	4 [0,1,3]	0 [0,0,0]	0 [0,0,0]	23 [11,9,3]	13.07% [15.49%,11.84%,10.34%
.5 Infeasible	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0% [0%0,%0,%]
.6 Inconsistent	1 [1,0,0]	13 [8,5,0]	1 [0,1,0]	3 [2,1,0]	5 [0,2,3]	0 [0,0,0]	0 [0,0,0]	23 [11,9,3]	13.07% [15.49%,11.84%,10.34%
.7 Over-specification	0 [0,0,0]	0 [0,0,0]	1 [0,1,0]	0 [0,0,0]	1 [0,0,1]	0 [0,0,0]	0 [0,0,0]	2 [0,1,1]	1.14% [0%,1.32%,3.45%]
.8 Not Traceable	0 [0,0,0]	1 [1,0,0]	2 [2,0,0]	1 [1,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	4 [4,0,0]	2.27% [5.63%,0%,0%]
.9 [Reserved for future]	0 [0,0,0]	1 [1,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	1 [1,0,0]	0.57% [1.41%,0%,0%]
.10 Non-Verifiable	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0% [0%,0%,0%]
.11 Misplaced	0 [0,0,0]	0 [0,0,0]	1 [1,0,0]	0 [0,0,0]	1 [0,0,1]	0 [0,0,0]	0 [0,0,0]	2 [1,0,1]	1.14% [1.41%,0%,3.45%]
.12 Intentional Deviation	2 [2,0,0]	1 [0,1,0]	1 [0,1,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	0 [0,0,0]	4 [2,2,0]	2.27% [2.82%, 2.63%,0%]
.13 Redundant or Duplicate	0 [0,0,0]	1 [1,0,0]	0 [0,0,0]	1 [0,0,1]	2 [2,0,0]	0 [0,0,0]	0 [0,0,0]	4 [3,0,1]	2.27% [4.23%,0%,3.45%]
Total	7 [7,0,0]	76 [32,43,1]	24 [8,16,0]	25 [16,5,4]	33 [5,6,22]	10 [2,6,2]	1 [1,0,0]	176 [71,76,29]	100.01% [100.01%,100%,99.99%
Percentage of ISS Faults by Year	3.98% [9.86%, 0%,0%]	43.18% [45.07%, 56.58%,3.45%]	13.63% [11.27%, 21.05%,0%]	14.20% [22.54%, 6.58%,13.79%]	18.75% [7.04%, 7.89%,75.86%]	5.68% [2.82%, 7.89%,6.90%]	0.57% [1.41%, 0%,0%]	99.99% [100.01%, 99.99%,100%]	

of CI B and CI C was planned for this year, but ended up being pushed back until late 2000 and late 2001 respectively due to hardware interface testing and development issues. The formal testing for the first release of CI A, originally planned for early 1998 and then the middle of 1998 and then the end of 1998, was completed between late 1999 and early 2000.

In 2000, in addition to the milestones mentioned above, an additional release of CI B was identified due to problems found during hardware integration and software stage testing. CI A release 2 underwent qualification testing in October 2000, delayed from late 1998 and again from late 1999. Also, final design reviews were held for CI C as well as software/hardware integration testing.

In July 2001, CI A underwent initial testing of the second version of release 2, delayed from earlier in the year due to functional changes. Also, the requirements for release 3 of this CI were established. Qualification testing of the rescaled second release of CI B began in September and was completed in November. Also, CI A underwent additional testing of the second version of release 2. Qualification testing for release 3, originally planned for early 2002, occurred in the middle of 2002.

Later in 2002, the requirements for release 4 of CI A were baselined. Qualification testing, planned for late 2002, was performed for this release in March 2003. The overall schedules of the flight software were continually changing during the development of the Station for a variety of reasons. The primary one relating to software requirements was the software ICDs. These ICDs were all bi-lateral between CIs. At the beginning of the software development effort, the development organization had not settled on a viable way to identify the interface requirements, mainly due to a lack of a good way to structure the requirements. This led to several aborted methods of documentation (including all electronic) as several different methods were attempted. For some CIs, the final solution did not arrive until after their first release of the requirements documents. This caused some rework and compromises in the designs. The final solution to document the interface requirements around communications protocols (ISO 7 layer model) and to restrict the data and command definitions to only data and commands actually used (functional rather than pass-through) finally resolved many issues.

4. Analysis of International Space Station Requirement Fault Frequency

The CI-process discussed in Section 2 was applied to the three CIs of the ISS. In addition to examining the requirement fault frequency by CI, the frequency was also examined by year. The sampling methodology used, trend analysis, discussion of findings, and lessons learned are presented below.

4.1 Sampling

The sampling was performed as follows. All requirement faults (documented in PRs for the ISS project) for each of the three CIs were extracted from the ISS configuration management tool and were saved in separate spreadsheets. Two senior verification and validation analysts (who are experienced with NASA software systems as well as other mission critical systems) were given the three spreadsheets and a set number of hours for the tasking (102 hours). The first analyst spent about 15% of his time formatting the data to ease categorization and calculations (roughly eight hours). He then spent one-third of his remaining hours (15.4 hours) on each of the three spreadsheets to ensure equal sampling (46.4 hours total). For each spreadsheet, he examined every fifth fault report and categorized it. Both the problem report description and the analysis report (written by the engineer who investigated the problem report) were used for categorization. He then gave a list of the problem report IDs that he had categorized to the second analyst (but not the categories that he assigned). The second analyst then categorized the same fault reports as the first analyst (he expended 47.25 hours on categorization). The analysts then met and came to agreement on fault categories for PRs with conflicting categories.

There are approximately 6,500 fault reports relating to requirements for ISS. There were 573 for the three CIs being examined. Of these, we examined 115, or 20%. Of those examined, we found that many were not requirement faults or were enhancement requests or suggested changes to process activities. These were marked as not applicable. Many of the problem reports documented more than one fault. We found that there were 176 faults within the fault reports that we examined, or on average 1.5 faults per problem report.

4.2 Trend Analysis and Discussion

Trend analysis was performed in two steps. First, visual analysis was performed and results were discussed with and reviewed by NASA engineers. In addition, using the year as a dependent variable and examining two independent variables (CI and fault category), analysis of variance was applied (assuming that the data is distributed normally). Our null hypotheses are that: there is no difference between the fault frequencies observed for the three CIs; there is no difference in the fault frequency observed for the seven years; and that there is no interaction between CI and year with respect to fault frequency. The alternative

Table 4. Analysis of variance results for all fault categories and CIs by year.

YEAR	SOURCE OF VARIATION	SS	DF	MS	F	P-VALUE	F CRIT	SIGNIFICANT?
1998	Fault category	2	4	0.5	1	0.46	2.8	no
1998	CI	3.33	2	1.67	3.33	0.088	3.11	yes, at 0.1 level
1999	Fault category	44.25	3	14.75	0.85	0.51	4.76	no
1999	CI	162.67	2	81.33	4.69	0.059	5.14	yes, at 0.1 level
2000	Fault category	6.27	4	1.57	1.92	0.2	2.8	no
2000	CI	20.13	2	10.06	12.33	0.003	3.11	yes, at 0.1 level
2001	Fault category	11.06	4	2.77	0.85	0.53	2.8	no
2001	CI	16.53	2	8.27	2.53	0.14	3.11	no
2002	Fault category	4.93	4	1.23	1.08	0.42	2.8	no
2002	CI	32.93	2	16.47	14.53	0.002	3.11	yes, at 0.1 level
2003	Fault category	7.33	4	1.83	3.79	0.05	2.8	yes, at 0.1 level
2003	CI	2.13	2	1.06	2.20	0.17	3.11	no
2004	Fault category	0.27	4	0.06	1	0.46	2.8	no
2004	CI	0.13	2	0.06	1	0.40	3.11	no

Table 5. 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

hypotheses are that differences do exist for the above. We will reject the null hypothesis in favor of the alternative hypothesis when the probability that the observed result is due to chance is 0.1 or less (alpha = 0.1). This alpha level is appropriate as we have a fairly small sample.

The results of these two steps are presented below, as well as a discussion of the findings. Table 3 provides the number of faults per year for all the CIs as well as the total for all three CIs. For example, reading across the row, in 1998, there was one total incompleteness fault across all three CIs; that fault was for CI A. In addition, incompleteness faults accounted for 23.3% of all faults, but accounted for 13.07% of the faults for CI A, for 30.26% for CI B, and for 31.03% for CI C. Reading down the 1998 column, there were a total of seven requirement faults found that year for the three CIs, or 3.98% of the total and 9.86% of CI A's faults.

4.2.1 Results by Year. In examining the results by year, we can see from Table 3 that 1999 had over twice as many requirement faults (43.18%) as any other year. Table 4 provides the sum of squares, degrees of freedom, mean of the squares, f-value, p-value, f-critical, and significance for all years, fault categories, and CIs. We can see from Table 4 that fault category is not significant for 1999 (p-value of 0.51), but that CI is statistically significant at the 0.1 level (p-value of 0.059). The next highest year for faults was 2002 at 18.75%. Also, configuration item C had five times more requirement faults in 2002 than for any other year. This was surprising to the NASA engineers. They believed that one reason for this might be that the requirements are now being written at a greater level of detail, and this might have previously lead to more occurrences of fault types 1.1 (incomplete) and 1.4 (ambiguous).

The year 1999 for configuration item A accounted for four-fold more faults than the year 2000 and two-fold more faults than the year 2001. This was not a surprise, as formal testing was occurring in late 1999, and one would expect the engineers to discover some requirements problems as they test the software (e.g., a test case fails, the ensuing debugging investigation indicates that the code and design were in accordance with the original requirement but that the requirement was faulty). Configuration item B also had many more faults in 1999 than the other years – almost three times as many as for the year 2000, and seven times as many as for the years 2001, 2002, and 2003. Note that the only remaining years (1998 and 2004) had no faults reported. This was not a surprise to the NASA engineers as 1999 was a year of much requirement work for CI B.

4.2.2 Results by CI. In Table 3, CIs A and B had very similar numbers of faults, with CI C having much fewer faults (29 total versus 71 and 76 respectively for CIs A and B). This was not surprising to the NASA engineers, and they felt it was not due to the requirements for CI C being of higher quality than for the other two CIs. This difference is explained in that the requirements were at a lower level of detail than the other CIs. This is the opposite effect of that mentioned in 4.2.1.

4.2.3 Results by Fault Category. Examining the summary of all three CIs (Table 3), one finds that categories 1.3 (incorrect), 1.1 (incomplete), 1.4 (ambiguous), 1.6 (inconsistent), and 1.2 (omitted) are dominant (in that order). In fact, for each of the three CIs, these five categories account for at least 85% of all the faults. In CI A, category 1.3 (incorrect) was by far the largest category at 30.99%. 1.4 (ambiguous) and 1.6 (inconsistent) represented half as many faults at 15.49%. CI B was dominated by categories 1.1 (incomplete) and 1.3 (incorrect) at 30.26% and 31.58% respectively. CI C had 1.1 and 1.3 as dominant categories at 31.03% and 24.14% respectively. It is interesting to note that category 1.1 (incomplete) was not as frequent for CI A as for the other two CIs, having only 12.68% of its faults in this category (as opposed to 30.26% and 31.03% for the other two CIs). The NASA engineers feel that this can be attributed to the system team writing the requirements for the CI A software. In this particular case, the CI A systems team was embedded in the development team. Therefore, less incomplete requirements existed for that CI than for the

Table 6. Analysis of variance results for CI by year.

SOURCE OF VARIATION	SS	DF	MS	F	P-VALUE	F CRIT	SIGNIFICANT?
Categories 1.1, 1.2, 1.3, 1.4, 1.6 by year	228.6	6	38.1	12.76236	3.44E-10	1.84	Yes, at 0.1 level
CI	33.66	2	16.83	5.636364	0.005	2.36	Yes, at 0.1 level
Interaction	217.1	12	18.1	6.060606	1.59E-07	1.63	Yes, at 0.1 level

other CIs that did not have as great an involvement from the system teams. Based on these results, we developed a tailored taxonomy for the three CIs (it is common to all three) by removing two fault types from the ISS taxonomy (infeasible and non-verifiable). This is shown in Table 5.

4.2.4 Interaction between CI and Year. We also examined the "within" effect of the independent variables. Table 6 provides the same columns as Table 4, but examines the five main fault categories (1.1, 1.2, 1.3, 1.4, 1.6) by year and possible interaction with CI. The table indicates that there is an interaction of CI and time (year) with a p-value of 1.59E-07.

5. Lessons Learned

As indicated above, each of the results was reviewed with a senior NASA engineer who has worked on the ISS program for 17 years. His question responses were then reviewed by several other experienced NASA engineers. The consensus opinion is reported here. In addition, the following questions were explored:

[1] Do you agree that the prominent fault categories shown in this study have also been "problem areas" historically? If so, why?

[2] Have steps already been taken in some of the non-prominent fault areas that may explain why those fault categories are so much lower?

[3] Are the prominent fault categories being addressed? Are there obvious common causes that point to ways to address these areas?

[4] Are there lessons learned as a result of this case history that can be used to help write better requirements in the future?

The engineers did agree that the five prominent categories (categories 1.3 (incorrect), 1.1 (incomplete), 1.4 (ambiguous), 1.6 (inconsistent), and 1.2 (omitted)) have historically been "problem" areas (question 1). They also noted that these were the same problems seen most frequently in requirements on other programs on which they have worked (ground real time, laboratory control, environmental control software systems). It has also been the experience of the first author that these same faults occur most frequently on the domains on which she has performed IV&V (specifically for weapon system software). Further discussion of why these have been historical problem areas will be presented below.

The engineers felt that fault frequencies in categories 1.8 (not traceable) and 1.10 (non-verifiable) were low because there are good tracing practices in place and these items are audited (question 2). Also, these fault types are easy to catch via audit. It was agreed that categories 1.11 (misplaced) and 1.12 (intentional deviation) should be expected to be low. If decomposition of the system requirements into elements and then major components (including the software) has been accomplished, with strong software ICDs between the CIs, then requirements should not be misplaced. The writing of intentionally deviating requirements is expected to be rare or non-existent.

The frequency of occurrence for category 1.13 (redundant or duplicate) faults was surprising to the engineers, in contrast with the first author's experience in her IV&V work (question 2). The engineers felt that since redundancy is designed into the CIs of the ISS, an engineer would rarely make the mistake of writing a redundant requirement. This may be a useful observation: that when engineers are specifically concentrating on writing requirements to ensure redundancy in a system, they will be less likely to introduce redundant requirements.

Engineers felt that category 1.5 (infeasible) faults were not frequently encountered for two reasons. First, many knowledgeable system engineers are involved in the requirements discussions and will discover potentially infeasible requirement ideas before they are written down. Second, the good traceability practices mentioned above, including tracing requirements to test cases early in the lifecycle (at the preliminary design review), ensure that any infeasible requirement will be detected very early in the lifecycle. Similarly, the NASA engineers felt that faults in category 1.7 (over-specification) would be rare because of the senior system engineers involved in the requirements discussions.

The engineers felt that several of the prominent fault categories could be explained by one phenomenon (question 3): 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.

Category 1.6 (inconsistent) was more perplexing. Though this category was not ranked as high as many of the others, it still accounted for roughly 10% of the faults. Interestingly, these faults were evenly split between external and internal inconsistencies (for every CI as well as for the roll up of all CIs). The engineers felt that there were no communication problems that contributed to this, as the same company worked on the requirements for each CI and the requirements and development software engineers even sat in the same area. One possibility is that the complexity of the domain of ISS and the detail level that is required in this type of a system makes this area a challenge, regardless of the skill level of the personnel or close communications. This is also an area for further study. We are currently expanding our common cause tree to more formally investigate the common causes for the most prominent fault categories.

The engineers noted that two of the CIs had very similar fault profiles, as expected. This was expected because they control multiple spacecraft systems (power, thermal, etc.) whereas the other CI does not. Also of particular interest is the data that the engineers thought they would see in the case history, but did not. For example, there was no perceivable difference seen in the number of faults occurring in a CI that was being developed by multiple companies using different internal processes. The engineers had hypothesized that this difference might have caused different fault profiles to occur; however, as noted above, the major difference had to do with the structure of the CI rather than the development culture.

The engineers felt that the immediate "lessons learned" from the case history were (question 4):

[1] Use systems engineers, responsible for the spacecraft systems being controlled by the software, when possible to assist with requirement specification,
[2] Use senior personnel, when possible, to specify requirements and to participate in requirement discussions,
[3] Continue with traceability activities,
[4] Have engineers document only one fault in a problem report,
[5] Further investigate the area of inconsistency,
[6] Identify processes which could be used to reduce the most frequently occurring fault types, and
[7] Document interface requirements around communications protocols and restrict data and command definitions to functional definitions used.

6. Future Work

Though we have made progress in this effort, much work remains. We have defined the future work in two phases. In the first phase, we will research existing IV&V technique taxonomies. Working with the NASA research community and one or more NASA projects, we will implement the process developed in [5, 6] to extend the IV&V techniques taxonomy. We will perform a literature survey for evidence that IV&V techniques detect certain requirements faults; build a traceability matrix (which techniques can detect fault types); use expert opinion to fill in gaps in the matrix; working with the Jet Propulsion Laboratory, populate the Advanced Risk Reduction Tool (ARTT) with this data; use expert opinion to validate the ARTT data (using different experts than for the matrix completion); and disseminate the findings.

As a second phase, we plan to examine the notion that projects specifically built with redundancy may not encounter redundant requirement faults. We plan to investigate the inconsitency fault category levels as well as perform common cause analysis. Additionally, examination of fault counting approaches for problem reports may provide interesting insights.

Elimination of requirement faults represents our greatest opportunity to save development cost and time. Thus we have pursued this first. Similarly, elimination of design faults is desirable. To that end, future work beyond this will concentrate on design techniques and faults, coding techniques and faults, etc. using the same approach for fault-based analysis that was used for requirements [5].

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 and Olga Dekhtyar for their assistance.

7. References

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

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

[3] 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.

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

[5] Hayes, J. Huffman. "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.

[6] Hayes, J. Huffman, SAIC, D.N. American. Final Report for Fault-Based Analysis: Improving Independent Verification and Validation (IV & V) through Requirements Risk Reduction. SAIC-NASA-98028. 20 December 2002.

[7] Helmer, G., Wong, J., Slagell, M., Honaar, V., Miller, L., and Lutz, R. ``A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System," Symposium on Requirements Engineering for Information Security (SREIS'01), March 5-6, 2001, Indianapolis, Indiana, Postscript. Extended version invited for Special Issue of The Requirements Engineering Journal, to appear.

[8] 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).

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

[10] 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.

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

[12] 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.

[13] Miller, L.A., Groundwater, E.H., Hayes J.E, Mirsky, S.M., "Guidelines for the Verification and Validation of Expert System Software and Conventional Software," NUREG/CR-6316, SAIC-95/1028, Volume 1.

[14] 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 (though subsequently cancelled in August 1998, this standard is still quite useful).

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

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

[17] 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.

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

[19] NASA Software Safety Guidebook. MIL-STD-882C.

[20] NASA Space Link, http://spacelink.nasa.gov/NASA.Projects/Human.Exploration.and.Development.of.Space/Human.Space.Flight/International.Space.Station/.

[21] Offutt, J. and Hayes, J. Huffman. "A Semantic Model of Program Faults," published in The Proceedings of the International Symposium on Software Testing and Analysis, pages 195-200, ACM, San Diego, California, January 1996.

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

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

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