Building a Requirement Fault Taxonomy: Experiences from a NASA Verification and Validation Research Project*

Jane Huffman Hayes
Computer Science Department
Lab for Advanced Networking
University of Kentucky
hayes@cs.uky.edu

EXPERIENCE REPORT

Abstract

Fault-based analysis is an early lifecycle approach to improving software quality by preventing and/or detecting pre-specified classes of faults prior to implementation. It assists in the selection of verification and validation techniques that can be applied in order to reduce risk. This paper presents our methodology for requirements-based fault analysis and its application to National Aeronautics and Space Administration (NASA) projects. The ideas presented are general enough to be applied immediately to the development of any software system. We built a NASA-specific requirement fault taxonomy and processes for tailoring the taxonomy to a class of software projects or to a specific project. We examined requirement faults for six systems, including the International Space Station (ISS), and enhanced the taxonomy and processes. The developed processes, preliminary tailored taxonomies for Critical/Catastrophic High-Risk (CCHR) systems, preliminary fault occurrence data for the ISS project, and lessons learned are presented and discussed.

1. Introduction

Verification and validation (V&V) activities are a subset of the overall development process and 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 is one way to approach these challenges.

Fault-based testing generates test data to demonstrate the absence of a set of pre-specified faults. Similarly, 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 project manager can use historical data to determine what fault types are most likely to be introduced or can perform a risk analysis to determine what fault types 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. Fault-based analysis can be used to improve the efficiency of the V&V or IV&V effort for any software development effort, though it is most commonly applied to development of CCHR systems. Note that fault-based analysis is an early lifecycle approach that can be applied prior to implementation.

One form of fault-based analysis, focused on requirement faults, can help us prevent and/or detect faults prior to developing design or writing code. A requirement fault is a fault that originates in the requirements phase (e.g., omitted requirement, incomplete requirement description). Eliminating faults as early as possible in the software lifecycle results in significant cost savings [1]. To provide a requirements-based fault analysis approach, we have defined an overall methodology [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.

The first phase of the research involved several objectives: select a fault taxonomy as the basis for the work, examine requirements faults from various projects, adopt or build a method for extending the fault taxonomy, and implement the method to extend the fault

* This work is supported by NASA under contract NAS2-98028.

taxonomy. In this paper, we present our methodology and application of its first phase at NASA.

The paper is organized as follows. Section 2 discusses related work in fault-based analysis. Section 3 describes our work and the results of the aforementioned research activities. Section 4 discusses the results of applying our fault taxonomy extension process to requirement fault data provided by the International Space Station. Section 5 presents a discussion of some lessons that we learned as we performed the research activities. Finally, conclusions and future work are presented in Section 6.

2. Related work

It is first necessary to define some of the terms that will be used in this paper. The IEEE standard definition of an error is a mistake made by a developer. An error may lead to one or more faults [7]. A Requirement Fault is a fault that originates in the requirements phase (e.g., omitted requirement, incomplete requirement description). We define Fault Taxonomy as an orderly classification of software faults according to their characteristics and relationships. We define Requirements Analysis as analysis of requirements to ensure completeness, consistency, clarity, explicitness, etc. [3]. A NASA Software Class is one of four possible classes (A through D) based on the NASA software classification scheme that considers the combined factors of cost, size, complexity, lifespan, risk, and consequences of failure. For each class there is a corresponding set of minimum requirements for software management, assurance, and engineering activities. This scheme is detailed in the NASA Software Safety Standard [15]. Class-Specific Taxonomy defines a taxonomy specific to a NASA software project Class. We introduce a process (termed Class-process) for developing this taxonomy. Project-Specific Taxonomy defines a taxonomy specific to a NASA project. We introduce a process (termed Project-process) that utilizes the results of Class-process as well as additional project-specific information and results in a project-specific taxonomy.

There are numerous fault-based testing techniques. These use a list of potential faults to generate test cases, generally for unit- and integration-level testing [13,2]. Research has been performed in the area of software safety fault identification also [12]. This includes research into numerous fault analysis techniques such as petri-net safety analysis [8], Failure Mode, Effects, Criticality Analysis (FMECA) [11], and criticality analysis [20]. A detailed literature survey into fault analysis techniques was performed and over 60 references are presented in [4].

As mentioned above, fault-based analysis has similarities to fault-based testing, where we target the strongest fault class while designing test generation algorithms in order to increase the effectiveness of the tests without unduly introducing overlap [6]. It is also risk driven, and attempts to select V&V techniques for application in order to achieve a project's goals. In this way, it is similar to test case selection during regression testing, where we attempt to reduce the time required to retest a modified program by selecting some subset of the existing test suite [18].

von Mayrhauser, Ohlsson, and Wohlin [19] determine what components are fault-prone using historical defect data and coupling and cohesion measures and derives related fault architectures. Our approach uses historical requirement defect data to understand what types of requirement faults are most prevalent for a particular NASA project or class of projects. Lutz [9] examines the root cause of software errors in embedded, safety-critical systems and identifies methods by which safety-related requirements errors can be prevented. Helmer et al [5] use a software fault tree for requirement identification and analysis in an intrusion detection system. We do not use a formal fault-tree analysis method to categorize our requirement defects, but we do consider the "root" cause when it appears that a fault might belong in more than one category.

Orthogonal Defect Classification, or ODC, classifies defects into non-overlapping categories and also examines defect triggers to lead to improvements in the software process [21]. ODC emphasizes design and code whereas our approach emphasizes requirements. ODC uses a fixed set of trigger and defect types. Our approach is based on tailored fault taxonomies. Munson and Nikora [14] count the number of faults associated with a failure using language-specific tokenizing. We have also found it difficult to "count" requirement faults. Often, more than one fault is described in a single Problem Report. We have performed some preliminary work in the area of fault "counting" and feel that this area requires further research [17].

Our research goes beyond that mentioned above in several ways. It aims to develop a general approach to fault-based analysis that can be applied to any software system development. To date, we have focused on requirement-based analysis and on NASA software systems. We have examined NASA requirement faults and have developed a generic taxonomy of these faults. One of our larger research goals focuses on V&V and IV&V techniques and their requirements fault detection/prevention capabilities. In future phases, we will work specifically to identify those techniques that will help reduce the requirements risks of NASA projects. We have identified new types of faults that

were not in existing fault taxonomies. Also, it is possible that our work will identify faults that are not detected (or not easily) by existing techniques (meaning that an outgrowth of this work may be the development of new or improved techniques).

3. Fault-based analysis at NASA

The problem addressed by this research is that there is never enough time or money to perform V&V on everything associated with a software project. We have only high-level knowledge of how the potential existence of specific requirements faults increases the risk of software projects. We have only high-level knowledge of how specific V&V techniques (requirements tracing, code analysis, etc.) contribute to improved system software reliability and reduced risk.

Therefore, there is a need to wisely select techniques to apply when performing V&V or IV&V on software programs. Resources are constrained, and we seek to lower program risk as much as possible with the least expenditure of time and money as possible. Specifically, we need to improve how we focus our resources for IV&V of Critical/Catastrophic High-Risk (CCHR) software functions. The nuclear power industry has found that a fault-based analysis method results in the optimal application of resources to V&V and IV&V of their critical software applications. They have identified the types of faults that are common in nuclear power system software requirements, and then have identified the requirements analysis techniques that can best prevent or detect these types of requirements faults [10]. This research categorized software systems as Class I, II, or III, where a malfunction of a Class I system could result in loss of life.

For the first phase of the research, the objectives were to: select a fault taxonomy as the basis for the work, examine NASA-specific requirements faults, build a list of IV&V techniques, adopt or build a method for extending the fault taxonomy, and implement the method to extend the fault taxonomy. Each will be discussed below.

3.1 Selecting a fault taxonomy

We chose the Nuclear Regulatory Commission (NRC) requirement fault taxonomy from NUREG/CR-6316 [10] as the basis for our work. We selected this taxonomy based on two key criteria:

- The fault categories were mutually exclusive, and
- The fault categories were not specific to a particular language, environment, or system development approach.

Also, as one of the developers of this taxonomy, the author was very familiar with it. The NUREG fault categories are listed below (subcategories can be seen in [10]):

1. Requirement Faults
   - 1.1 Incomplete decomposition
   - 1.2 Omitted requirement
   - 1.3 Improper translation
   - 1.4 Operational environment incompatibility
   - 1.5 Incomplete requirement description
   - 1.6 Infeasible requirement
   - 1.7 Conflicting requirement
   - 1.8 Incorrect assignment of resources
   - 1.9 Conflicting inter-system specification
   - 1.10 Incorrect or missing external constants
   - 1.11 Incorrect or missing description of initial system state
   - 1.12 Over-specification of requirements
   - 1.13 Incorrect input or output descriptions

We found many papers that confirmed our requirements fault types and found only a few papers that described "new" requirement faults (see [4]). Note that the above list, with the addition of not traceable, non-verifiable, unachievable, misplaced, and intentional deviation fault types, should serve as a good starting taxonomy for the fault-based analysis of any software system. Examination of NASA requirement faults resulted in a number of changes to the taxonomy, as discussed below. The resulting new set of 13 fault categories was considered to be relevant as a "generic" NASA fault taxonomy as shown in Table 1.

We added a category for each "new" fault type such as not traceable, non-verifiable, unachievable, misplaced, and intentional deviation, bringing the taxonomy to 18 fault types. We consider **omitted** or **missing** requirement as one major category. The sub-fault categories identified under this category are: 1) Omitted requirement, 2) Missing external constants, and 3) Missing description of initial system state. We identified **incorrect** as one major category. The sub-fault categories are: 1) Incorrect external constants, 2) Incorrect input or output descriptions, 3) Incorrect description of initial system state, and 4) Incorrect assignment of resources.

We added one major requirement fault, **ambiguous,** and under it we grouped: 1) Improper translation, and added a new sub-fault category 2) Lack of clarity. We grouped **conflicting** requirements into one major fault category, **inconsistent,** and the sub-faults under this category are: 1) External conflicts, and 2) Internal conflicts. We added a new major requirement fault, **redundant**, to cover the situation where a requirement appears duplicated elsewhere in the specification. We

Table 1. Revised requirement fault taxonomy.

Major Fault	Sub-Faults	Description of Sub-Faults	Earlier version taxonomy faults that are mapped here
1. Requirements	riginate in Requirements phase; found in the Requirements Specification		
.1 Incompleteness	.1.1 Incomplete Decomposition .1.2 Incomplete Requirement Description	.1.1 Failure to adequately decompose a more abstract specification. .1.2 Failure to fully describe all requirements of a function.	.1,.5
.2 Omitted/Missing	.2.1 Omitted Requirement .2.2 Missing External Constants .2.3 Missing Description of Initial System State	.2.1 Failure to specify one or more of the next lower levels of abstraction of a higher level specified. .2.2 Specification of a Missing value or variable in a requirement. .2.3 Failure to specify the initial system state, when that state is not equal to 0.	.2,.10,.11
.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	.3.1 Specification of an incorrect value or variable in a requirement. .3.2 Failure to fully describe system input or output. .3.3 Failure to specify the initial system state, when that state is not equal to 0. .3.4 Over-or-under stating the computing resources assigned to a specification.	.10,.11, .13, .8
.4 Ambiguous	.4.1 Improper Translation .4.2 Lack of Clarity	.4.1 Failure to carry detailed requirement through decomposition process, resulting in ambiguity in the specification. .4.2 difficult to understand or lack of clarity and therefore ambiguous.	.3
.5 Infeasible	. ----------	.5.1 Requirement, which is unfeasible or impossible to achieve given other system factors, e.g., process speed, memory available.	.6
.6 Inconsistent	.6.1 External Conflicts .6.2 Internal Conflicts	.6.1 Requirements that are pair-wise incompatible. .6.2 Requirements of cooperating systems, or parent/embedded systems, which taken pair-wise are incompatible.	.7,.9
.7 Over-specification	----------	.7.1 Requirements or specification limits that are excessive for the operational need, causing additional system cost.	.12
.8 Not Traceable	.----------	.8.1 Requirement which cannot be traced to previous or subsequent phases.	.14
.9 Unachievable Item	----------	.9.1 Requirement that is specified but difficult to achieve. The requirement statement or functional description cannot be true in the reasonable lifetime of the product.	.15
.10 Non-Verifiable	----------	.10.1 The Requirement statement or functional description cannot be verified by any reasonable testing methods Process exists to test satisfaction of each requirement. Every requirement is specified behaviorally.	.16
.11 Misplaced	----------	.11.1 Information which is in a different section in requirements document.	.17
.12 Intentional Deviation	----------	.12.1 The Requirement that is specified at higher level but intentionally deviated at lower level from specifications.	.18
.13 Redundant or Duplicate	----------	.13.1 Requirement was already specified elsewhere in the specification	----------

left the remaining requirement faults as one major category as each of these fault types do not overlap.
Though the NUREG taxonomy included an **operational environment incompatibility** category, there is no such category in our revised requirement fault taxonomy. This is because the requirement sub-fault **missing external constants** subsumes operational environment incompatibility fault. This is a more detailed or decomposed lower level fault of missing external constants and we found that it is very difficult to make a clear distinction between these two faults during the requirements phase. In order to avoid overlap, we consider any fault dealing with operational environment incompatibility as the sub-fault **missing external constants** under the Omitted requirement major fault. The resulting taxonomy has 13 categories and can be seen in Table A1.

3.2 Examining requirements faults

Obtaining NASA project-specific fault data proved to be challenging. As one would expect, there is a required level of security for the manned space flight programs. Fault reports cannot be made publicly available, and appropriate research agreements must be put in place to ensure the security of the fault data while still allowing researchers some latitude for publication of results. The data was essential for the research. We received and examined IV&V "comments" on requirement problems for four projects and Project fault reports (requirements-related) for another two projects. We noted that the level of detail of the fault data provided varied greatly. Analysis of the data provided insights that allowed us to improve our generic fault taxonomy and our taxonomy extension/tailoring processes.

3.3 Process for extending the fault taxonomy

We have built and adopted a method for extending or tailoring the fault taxonomy, as mentioned in section 2. We split our process for extending the fault taxonomy into two parts: Class-process and Project-process. Class-process discusses all the activities that are to be performed to develop a class-specific taxonomy. The outputs of Class-process are inputs to the Project-process

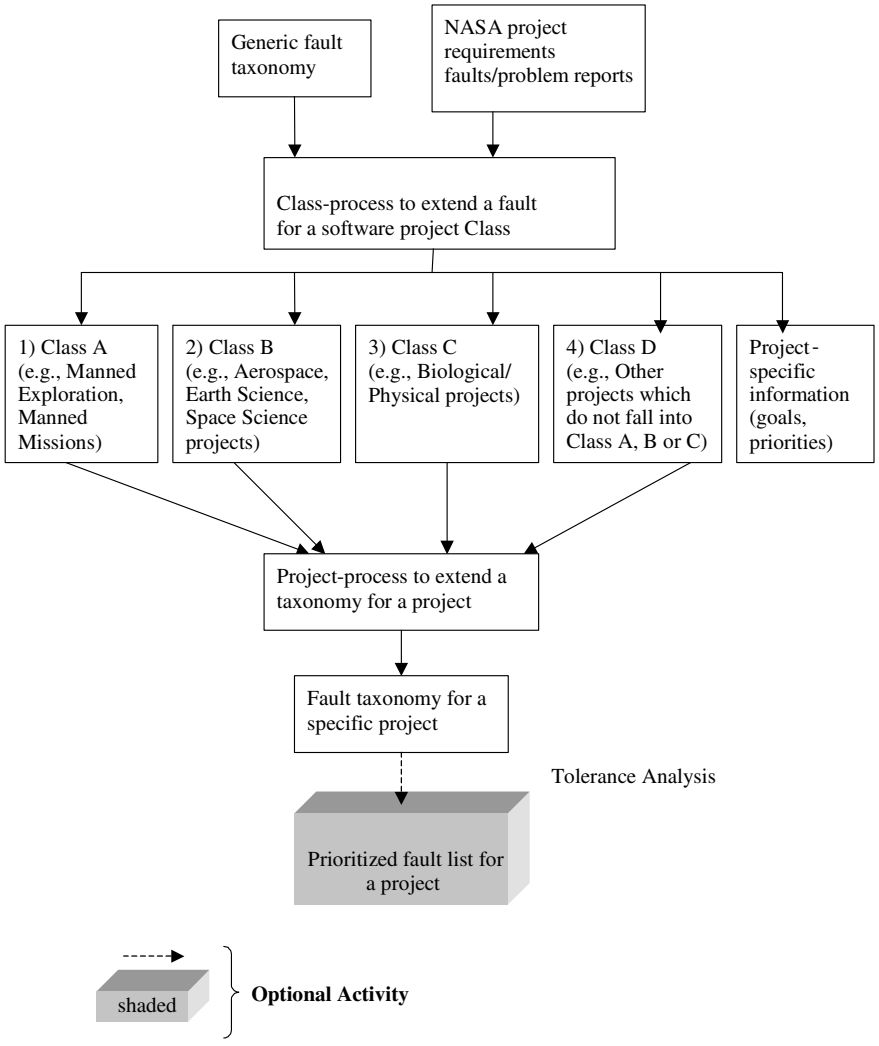

Figure 1. High-level process to extend a fault taxonomy.

(i.e., we take a class-specific taxonomy and perform all the activities described in the Project-process section to develop a project-specific taxonomy).

The Process for developing the class-specific requirement fault taxonomy is shown in Table 2. The table consists of six fields: entry criteria, activities, exit criteria, inputs, process controls and metrics, and outputs. The entry criteria field describes a checklist of pre-conditions that must be met before the process activities can start. All the information and data needed such as the generic fault taxonomy, NASA project requirement faults, problem reports and class project definitions must be available before the process starts. NASA must authorize the use of project data. In addition, it is necessary that NASA has authorized the taxonomy extension project.

Table 2. Class-process for extending a fault taxonomy for Classes (A-D) of NASA software projects.

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 generic requirement fault taxonomy 2. Examine problem reports for projects in Class A, B, C, and/or D 3. Categorize the faults for each project according to the generic taxonomy 4. Determine frequency fault types for each class and percent of fault occurrences 5. Identify crucial fault categories for each class	1. A Class-specific requirement fault taxonomy has been developed (Class A, B, C, and/or D)
Inputs	**Process Controls/Metrics**	**Outputs**
1. Generic fault taxonomy 2. NASA project requirement faults/problem reports 3. Class project definitions	Controls: 1. Maintenance of configuration control of taxonomy 2. Maintenance and management of NASA project data by class Metrics: 1. Person Hours of effort 2. # of projects 3. # faults 4. frequency of fault 5. % of fault occurrence 6. Top 5 Historical Fault areas by class	1. Frequency counts of faults per class and percent of fault occurrences 2. Crucial fault categories for each class

Our process for extending the fault taxonomy is shown in Figure 1. The process builds on our generic taxonomy. Just as NRC examined classes of software projects separately [10], we feel there will be a substantial difference in the proportion of faults between NASA software project classes. First, we take our generic fault taxonomy, NASA project requirement faults and problem reports and perform Class-process as discussed below. The result is a NASA software project class taxonomy. The criteria for the classes are found in the NASA Software Safety Guidebook [15]. We grouped manned missions and manned exploration projects into Class A, aerospace, earth space, and science space projects into Class B, biological and physical projects into Class C, and the remaining projects which do not satisfy any of the prior class conditions into Class D. Next, the Project-process is performed. The result is a project-specific requirement fault taxonomy. Finally, an optional activity is to perform tolerance analysis and to develop a prioritized

fault list for the project. Due to space limitations, only the Class-process will be discussed in this paper. The interested reader can see the NASA report [4] for more details on the Project-process and tolerance analysis.

The activities to be performed include selecting a generic requirement fault taxonomy, obtaining problem reports for projects in Class A, B, C, and D, categorizing the faults obtained for each project using our fault taxonomy, determining the number of faults for each category and the percentage of occurrences, and identifying the top five critical requirement faults for each Class A, B, C, and D.

We estimate fault frequency for different projects under each class. For example, we use a table to accumulate fault frequency for aerospace, earth science and space science projects under the Class B category. Then, we identify the requirement fault types, fault frequency count, and percentage of fault occurrences for

each project. Assume that 50 incomplete decomposition requirement faults exist in the Class B projects and that 10 incomplete description faults exist. Overall 1000 requirement faults were found for Class B. The percentage of occurrence of incomplete requirement faults is therefore 6% for Class B.[1]

Finally, we will determine the historically most probable requirement faults for each class. We list the top five major and sub-requirement faults for a Class of projects. We then assign a complexity of high, medium, or low depending upon a fault's frequency. If certain faults are found more frequently for a certain class, then it is crucial to seek improvement in that area and to attempt to prevent and/or detect these fault types.

The outputs of this process are the frequency counts of the faults, percent of fault occurrence, and the crucial requirement faults for each class. We repeat this process for each class for which we have project data until our exit criteria is met (i.e., we have developed a class-specific requirement fault taxonomy). The process controls ensure that all versions of our requirement fault taxonomy are properly maintained under configuration control. Also, the NASA project data must be maintained by project class. Process metrics include person hours for the effort, number of projects, number of requirements faults, etc.

3.4 Implementing the fault taxonomy extension process

This research activity entailed implementing the method to extend the fault taxonomy to develop a class-specific taxonomy (Class-process) and a project-specific taxonomy (Project-process). Prior to beginning the Class-process, feedback from staff at the IV&V Facility in West Virginia corroborated the researcher's categorization of one project as NASA Class C; two projects as NASA Class B, and the ISS project as NASA Class A. It is preferable to classify as many projects per class as possible. Unfortunately, these were all the projects for which we had data. We desired more project data, and would have certainly preferred more than one class A project and more than one Class C project.

Three verification and validation analysts independently examined and categorized project faults for the six data sources. On average, the analysts had eight years of V&V experience. Each analyst followed the fault taxonomy extension process for NASA software classes. During this process, lessons learned from each analyst resulted in revisions, clarifications, deletions, and additions to the generic taxonomy as well as to the process itself. Insight was also gained during a review of the orthogonality concept as applied to these taxonomy categories. Our conclusions on orthogonality are presented in Section 4. It should also be noted that the analysts only consulted with each other to verify that they were following the categorization process consistently and had a shared understanding of the generic taxonomy and associated category definitions.

It was noted that in many cases across the six project data sources, multiple requirement faults were included in a single Project Problem Report (PR) or IV&V comment. This warranted special attention by the analysts to properly count and categorize project fault data. Based on this observation, one suggestion is for the projects and/or reviewing IV&V analysts to document each individual fault separately.

The three analysts, in conjunction with the author, made the following changes to the revised generic taxonomy (as can be seen in Table 3):

- Descriptions of the several Fault and Sub-fault categories were clarified to reduce confusion among present and future analysts using this generic taxonomy. Descriptions now align closely with the intent of the category or subcategory. In some cases, elaborative comments or examples were added in the last column of the table. All of the fault category item descriptions were clarified except for Category 1.7.
- Mainly for reasons of orthogonality, the following categories or subcategories were combined due to their similarity with or indistinguishability from other categories or subcategories: Subcategories 1.1.1 and 1.1.2 were combined; Subcategories 1.4.1 and 1.4.2 were combined; and Categories 1.5 and 1.9 were combined. Category 1.9 is now "Reserved for future."
- The following subcategory was deleted/removed from the taxonomy, again due to orthogonality or similarity issues: Subcategory 1.2.3.

Upon completion of the Class-process, review of our analysts' combined categorization data for six projects across three NASA classes (A, B, and C) allowed us to develop three class-specific taxonomies. The class-specific taxonomy for Class A, a sub-set of the final generic taxonomy, is shown in Table 3. It should be noted that this is a first "draft" taxonomy, as we only looked at data for one project in this class. In the future we plan to repeat the Class-process for additional projects/systems in each class (including Class D) in order to refine and finalize class-specific taxonomies for each of the four NASA Classes.

[1] The author is aware that faults are not "created equally" and is currently working on advancing ideas from [17] on examining the relative semantic "size" of faults. In the meantime, one problem report is equivalent to one fault.

We were unable to completely implement the Project-process. This was due to logistical problems in obtaining project data, small fault sample sizes, lack of historical fault data for previous software versions, and the insufficient level of detail of the fault data provided by some of the NASA projects. Hence, we could not produce an accurate and validated project-specific taxonomy for these projects. However, we were able to perform the Class-process. For Class B we found that 83% of the faults fell under three categories: Incompleteness, Ambiguous, or Inconsistent. From the Class C data, we found that 83% of the faults fell under three categories: Incompleteness, Omitted, or Incorrect. We found that the NASA project managers were pleased to gain this insight into their programs. Specifically, they asked how they could improve the writing of requirement specifications to decrease or eliminate these categories of mistakes.

Table 3. Draft class-specific requirement fault taxonomy for NASA Class A projects.

Major Fault	Sub-Faults	Description of Sub-Faults
.1 Incompleteness	----------	.1.1 An abstract specification from a higher level document exists in a lower level document, but it has not been fully elaborated or expanded.
.2 Omitted/Missing	.2.1 Omitted Requirement	.2.1 An abstract specification or concept from a higher level document does not exist in or is completely omitted from a lower level document.
	.2.2 Missing External Constant	.2.2 Reference to an external constant that doesn't exist or the value of an external constant is not specified.
.3 Incorrect	.3.1 Incorrect External Constant	.3.1 Specification of an incorrect value or variable in a requirement that does not conflict with a cooperating or parent/embedded system.
	.3.2 Incorrect Description of Input or Output	.3.2 Incorrect description of system input or output.
	.3.3 Incorrect Description of Initial System State	.3.3 Incorrect description of initial system state, when that state is not equal to 0.
	.3.4 Incorrect Assignment of Resources	.3.4 Overstating or understating the computing resources assigned to a specification.
.4 Ambiguous	----------	.4.1 The requirement wording is difficult to understand due to poor grammar usage or word choice, resulting in misinterpretation or leading to multiple valid interpretations.
.5 Infeasible	----------	.5.1 Requirement that is specified, which is unfeasible and difficult or impossible to achieve given other system factors (e.g., process speed, memory available) and cannot be true in the reasonable lifetime of the product.
.6 Inconsistent	.6.1 Internal Conflicts	.6.1 Requirements that are pair-wise incompatible in same document or between requirement documents at the same level.
	.6.2 External Conflicts	.6.2 Requirements of cooperating systems, or parent/embedded systems, which taken pair-wise are incompatible. Includes conflicting references to external functionality.
.7 Over-specification	----------	.7.1 Requirements or specification limits that are excessive for the operational need, causing additional system cost.
.8 Not Traceable	----------	.8.1 Requirement which is "out-of-the-blue" and cannot be traced to lower level or higher level specifications.
.9 [Reserved for future]	----------	----------

4. International Space Station

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 [16]. Just as the Phase I research funding was expiring, we established a research agreement with the International Space Station (ISS). The project provided 6500+ Problem Reports (PRs), which we estimate described as many as 8500+ requirement faults (many individual PRs described multiple requirement faults). ISS Fault data contained greater detail than we had seen for the Class B and C projects. We were able to examine 10% of the project fault data. Of the 10%, a fair number were not requirement faults and our final sample size of ISS requirement faults (486 faults) only represented 6% of all the fault data provided.

As discussed above, we performed the Class-process for Class A systems (even though we only had data from one project). We began to perform the Project-process. It became apparent that to accurately develop and validate a project-specific taxonomy for the ISS project, we would still need additional information. For example, we would need: more detail on some of the fault descriptions provided; to examine the appropriate referenced documents (i.e., the actual requirement specifications); to analyze a larger sample size; to review historical fault data, if available; and to engage in brief clarification discussions with project staff. Hence, we did not complete the Project-process for the ISS project. The ISS project has expressed their desire to continue the research collaboration and we do plan to complete the Project-process in the near future.

Nonetheless, the classification analysis of the resulting 486 ISS requirement faults did yield some initial data as shown in Table 4. We presented the ISS data findings to the project in Houston, Texas in December, 2002. We discussed the top three significant taxonomy categories into which their requirement faults fit as well as the percentage of faults in those categories. These three fault categories and their percentages were: Incompleteness (20.9%), Omitted/Missing (32.9%), and Incorrect (23.9%). Together, these three categories accounted for almost 80% of the requirement faults evaluated. Project management stated that they found this data useful toward the development of future requirements.

5. Discussion

We learned a number of interesting things from this study. A lesson learned from the implementation of the Class-process had to do with re-examining the concept of orthogonality. It was agreed, between the analysts and researchers, that the average analyst using the taxonomy to categorize requirement faults might determine that more than one category applied to a single requirement fault. This stimulated a discussion as to whether a requirements fault taxonomy, or category within a taxonomy, could really be orthogonal. We determined, that to eliminate any confusion by users of this process and its taxonomy, that an analyst must dig down to find the root cause of a requirement fault. A single fault taxonomy category was then almost always apparent.

In many cases during our analysis, we found that we did not have access to all the information we needed to determine the root cause or to pick one sub-category over another (e.g., 1.6.1 versus 1.6.2). The Class-process has also been modified to guide the analyst to select the Major Fault Category when there is not enough information to select one of the subcategories (e.g., categorizing a fault as "inconsistent" Category 1.6 if one cannot select either 1.6.1 or 1.6.2). The information we would have liked to receive (e.g., high level parent specification, detailed children or interface specifications, Configuration Control Board meeting minutes or other documents explaining project changes to specifications) was not easily obtainable from the projects participating in this study. We concluded that data gathered by this approach could be improved through direct interaction with project personnel.

Table 4. ISS project categorization percentage data.

Major Fault	Percentage 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 or Duplicate	0.005

Even armed with enough information, it would seem reasonable, for example, that an analyst looking at a

requirement fault documenting a "missing" requirement (Category 1.2) might think the following: "If the requirement is missing or omitted, it is at the same time also untraceable (Category 1.8)". While this may seem true, the root cause of the fault is the omission. A requirement that appears "out of the blue" or unconnected to any other specification (child specification or same level specification, such as interface specification) is a root cause "not traceable" requirement (Category 1.8). We clarified Category 1.8's definition and definitions of the other categories to make it as clear as possible into which category the root cause of a fault fits. In sum, analysis of our taxonomy leads us to conjecture that it is largely orthogonal, although in a few instances multiple categories applied.

6. Conclusions and future work

In this work, we focused on building a taxonomy of requirement faults and a process for extending the taxonomy. By eliminating faults at the earliest possible opportunity, we reduce the cost of fixing errors later in the lifecycle [1]. The research, to date, has resulted in a number of general findings:

- A NASA specific taxonomy did not previously exist, is needed, and was developed.
- A process for tailoring/extending a taxonomy did not previously exist, is needed, and was developed.
- Sub-faults are useful for clarification only.
- Classification can best be done at the fault level.
- NASA Fault reports have varying levels of detail.
- NASA Problem Reports (PRs) often cover multiple faults.
- Source Codes of PRs (stating what phase of the lifecycle or what artifact resulted in the PR) are not always accurate.
- Data needed for performing the Project-process is considered sensitive and may be difficult to obtain.
- New PRs are often written when old problems are not fully corrected.
- It is not easy to determine into which class a NASA projects falls.
- Use of multiple analysts and projects enhanced both our taxonomy and our processes.

Based on our work and the findings above, we made the following recommendations to NASA, many of which may apply generally to all software projects:

- Publish a NASA specific requirement fault taxonomy.
- Publish a process for extending a taxonomy.
- Publish a "requirement writer's guide."
- Recommend to projects that they write cohesive PRs, with one fault per PR.
- Publish a list of NASA Projects by class.

Though we have made good progress in Phase I of this effort, much work remains to be done. We have defined the future work in two phases, Phase II and Phase III. Our approach for Phase II will be: research existing IV&V technique taxonomies; working with the NASA research community and one or more NASA projects, implement the process developed in Phase I to extend the IV&V techniques taxonomy (to fully cover NASA needs); perform a literature survey for evidence that IV&V techniques detect certain requirements faults; build a traceability matrix (what 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) implementation of DDP with this data; use expert opinion to validate the ARTT data (using different experts than for the matrix completion); and disseminate the findings. In Phase III, we propose to: design and implement an experimental pilot study [10] that has an IV&V agent and/or a NASA project use the process to extend a fault taxonomy for a specific project to assess its usefulness. We plan to have an IV&V agent use the populated ARTT on a Goddard Space Flight Center Code S project to validate the usefulness of our technique-to-defect mapping. We will also document our fault-based analysis technique and ARTT for broad use (to support technology transfer).

The ISS program has expressed interest in examining their requirement fault "profiles" over time. For example, what would Table 4 look like for requirement problem reports from January 1, 1999 – December 31, 2000, from January 1, 2000 – December 31, 2001, etc.? We hypothesize that the ISS fault taxonomy will remain fairly stable, but that it is possible that we may discover additional categories as we expand our examination of ISS problem reports. We agree with the ISS personnel that at different points in time, there may have been differences in the proportion of faults per category.

Elimination of requirement faults represents our greatest cost saving opportunity and 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 that was used for requirements.

7. Acknowledgments

Our work is funded by NASA under contract NAS2-98028. Our thanks to Ken McGill, Tim Menzies, Stephanie Ferguson, Pete Cerna, Mike Norris, Bill Gerstenmaier, Bill Panter, and the International Space Station project. We thank D.N. American, Bill Wyatt, and Pam Skotak, Jack Abraham, Leo Kotowski, and Rick Hallsworth of SAIC for their contribution to the project. Thanks to Professors Jeff Offutt and Gregg Rothermel for comments and suggestions that greatly improved this paper.

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

[4] 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.

[5] 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.

[6] 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).

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

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

[9] 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.

[10] 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.

[11] 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).

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

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

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

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

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

[17] 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.

[18] Rothermel, G., Harrold, M.J., Analyzing regression test selection techniques. IEEE Transactions on Software Engineering, 22(8), Aug. 1996.

[19] 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.

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

[21] 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.