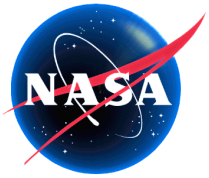
:

Final Report for Fault-Based Analysis: Improving Independent Verification and Validation (IV & V) through Requirements Risk Reduction

Contract Number: NAS2-98028
Document Number: SAIC-NASA-98028

20 December 2002

Prepared for:



National Aeronautics and Space Administration
Software IV&V Facility
100 University Drive
Fairmont, West Virginia 26554

Prepared by:

Dr. Jane Hayes, UKY/Science Applications International Corporation/D.N.American

Table of Contents

List of Tables

Executive Summary

The purpose of this research is to develop a requirements-driven method for selecting IV&V techniques to apply. The first phase of the work concentrated on building a NASA-specific fault taxonomy, as well as a process for tailoring it to a class of projects or a specific project. During this year of the project we selected a known requirement fault taxonomy, from NUREG/CR-6316 [4], and performed a thorough literature survey to enhance it (Task 1). We examined NASA requirement faults for 6 systems and enhanced the taxonomy further (Task 4). These activities resulted in a reorganization of and three additions to the taxonomy. We developed two processes, one for tailoring the NASA taxonomy for classes of software projects, and one for tailoring the taxonomy for a specific software project (Task 6). We then performed the first process several times, further improving the process each time (Task 7). The participation of the International Space Station (ISS) project was pivotal in this activity. We also developed a preliminary list of IV&V techniques that can be applied during the requirement phase (Task 5). We presented the results of our work at two Program Management Reviews (PMRs), at the NASA Software Assurance Symposium, and to the ISS project at JSC, Houston, Texas. The final PMR presentation is enclosed with this report (Section 3.9). Much positive feedback regarding this project has been received, particularly from the ISS project. Though the follow-on to this project has not been funded, we plan to pursue other funding avenues to continue the research.

Final Report for Fault-Based Analysis: Improving Independent Verification and Validation (IV & V) through Requirements Risk Reduction

1. Introduction

This final report presents the findings of the first phase of the project for Fault-Based Analysis: Improving Independent Verification and Validation (IV&V) through Requirements Risk Reduction.

The problem addressed by this project is that there is never enough time or money to perform IV&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 NASA projects. We have only high-level knowledge of how specific IV&V techniques (requirements tracing, code analysis, etc.) contribute to improved NASA system software reliability and reduced risk.

Therefore, there is a need to wisely select techniques to apply when performing IV&V on NASA 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.

The project objective in the first phase was to develop a requirements fault taxonomy for NASA systems (expandable to a general one), develop a taxonomy of IV&V techniques, examine requirements analysis techniques to determine what faults they can detect, and develop guidance for NASA IV&V based on the results. For the first phase of the project as detailed in this final report, the objectives were to build a list of IV&V techniques, adopt or build a method for extending taxonomies, and implement this method for a requirements fault taxonomy for NASA systems (expandable to a general one).

All first phase project objectives were met and the results per project task are detailed in this report. This project had nine specific tasks as follows:

- Task 1 – Select a Known Fault Taxonomy
- Task 2 – Presentation and Milestone Meeting 2 (PMR 2)
- Task 3 – Presentation and Milestone Meeting 3 (PMR 3)
- Task 4 – Examine NASA-specific requirements faults
- Task 5 – Build a list of IV&V techniques

- Task 6 – Adopt or build a method for extending the taxonomy
- Task 7 – Implement the method to extend the fault taxonomy (to fully consider NASA systems)
- Task 8 – Document all results
- Task 9 - Year-end report and presentation (PMR 4)

The organization of this report is as follows:

Section	Content
Section 1	Introduction
Section 2	Standard Definitions
Section 3	Results by Task (Tasks 1 – 9)
Section 4	Conclusions/Findings and Recommendations
Section 5	Future
Section 6	References

2. Standard Definitions

Fault – defect or flaw.

Fault Taxonomy – orderly classification of software faults according to their characteristics and relationships.

Requirements Analysis – analysis of requirements to ensure completeness, consistency, clarity, explicitness, etc. [1].

Requirement Fault – fault that originates in the requirements phase (e.g., omitted requirement, incomplete requirement description).

NASA Software Class – A NASA software classification scheme developed based on 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 [3].

Class-Specific Taxonomy – Starting with our researched generic fault taxonomy, NASA project requirement faults and problem reports, and project specific information, perform Process A as discussed in Section 3.6.3.3. The result is a taxonomy specific to a NASA software project Class.

Project-Specific Taxonomy – Starting with the appropriate class-specific taxonomy, NASA project requirement faults and problem reports, and project specific information, perform Process B as discussed in Section 3.6.3.4. The result is a taxonomy specific to a NASA project.

3. Results by Task

The next few sub-sections present the detailed results of each task associated with the first phase of this project.

3.1 Task 1 - Select a Known Fault Taxonomy

For this task, we first chose the Nuclear Regulatory Commission (NRC) requirement fault taxonomy from NUREG/CR-6316 [4], as shown in Table 3-1. We selected this taxonomy based on two key criteria:

1. The fault categories do not overlap with each other, and

2. The fault categories were not specific to a particular language, environment, or system development approach.

Next, we performed a thorough search for requirements fault taxonomies. We searched resources such as IEEE, ACM, Altavista, Hotbot, Google, Yahoo, Lycos, Excite, Wilson Web, Kluwer, Cambridge Scientific Abstracts (CSA), Cite seer Search engines, CERT Coordination Center (SEI), NASA website, NASA IV & V Facility online resources, and libraries. We found many papers that confirmed our requirements fault types and found only a few papers that described "new" requirement faults. Our literature survey encompassed 61 references. We added a category for each "new" fault type such as not traceable, non-verifiable, unachievable, misplaced, and intentional deviation. Based on this, the requirement fault taxonomy is shown in Table 3-2.

We identified 18 requirement faults as opposed to the 13 in Table 3-1. As we examined the faults very closely we found that there was an overlap in the existing taxonomy. As shown in Table 3-3, the taxonomy was trimmed to 13 main requirement fault categories instead of 18 categories in Table 3-2. These 13 are somewhat different from the original 13 in the NRC taxonomy we started with in Table 3-1. The resulting new set of 13 categories was considered to be relevant as a NASA "generic" fault taxonomy.

We arrived at 13 categories as follows. We grouped all incomplete requirement faults into one major category **incomplete**. Under incomplete major requirement fault, we have two subfault categories: 1) Incomplete Decomposition and 2) Incomplete Requirement Description. These are categories .1 and .5 in Table 3-2. We made a distinction between omitted/missing requirement and incorrect requirement by making categories .10 and .11 two separate major fault categories in Table 3-2.

We consider **omitted** or **missing** requirement as one major category. The subfault categories identified under this category are: 1) Omitted requirement, 2) Missing External Constants, and 3) Missing Description of initial system state. These are .2, .10, and .11 categories in Table 3-2. We identified **incorrect** as one major category. The subfault 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 (i.e., categories .10, .11, .13, .8 of Table 3-2).

We added one major requirement fault, **ambiguous,** and under it we grouped: 1) Improper translation (category .3 of Table 3-2), and added a new subfault category 2) Lack of clarity. We grouped **conflicting** requirements into one major fault category, **inconsistent,** and the subfaults under this category are: 1) External Conflicts, and 2) Internal Conflicts (i.e., .7 and .9 categories in Table 3-2). We added a new major requirement fault, **redundant**, to cover the situation where a requirement appears duplicated elsewhere in the specification. We left the remaining requirement faults as one major category as each of these fault types do not overlap.

As you can observe, there is no **operational environment incompatibility** category in our revised requirement fault taxonomy in Table 3-3. This is because the requirement subfault **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 under operational environment incompatibility category as subfault **missing external constants** under the omitted requirement major fault.

Table 3-1. NRC Requirement Fault Taxonomy [4].

Type	Description	Occurs
0. Requirements	Originate in Requirements phase; found in the Requirements Specification	
1. Incomplete Decomposition	Failure to adequately decompose a more abstract specification.	System, Sub, Mod
2. Omitted Requirement	Failure to specify one or more of the next lower levels of abstraction of a higher level specified.	System, Sub, Mod
3. Improper Translation	Failure to carry detailed requirement through decomposition process, resulting in ambiguity in the specification.	System, Sub, Mod
4. Operational Environment Incompatibility	Specification which does not accommodate the operational environment, such as data rates, data formats, etc.	System, Sub, Mod
5. Incomplete Requirement Description	Failure to fully describe all requirements of a function.	Mod
6. Infeasible Requirement	Requirement, which is unfeasible or impossible to achieve given other system factors, e.g., process speed, memory available.	Mod
7. Conflicting Requirement	Requirements that are pairwise incompatible.	System, Sub, Mod
8. Incorrect Assignment of Resources	Over-or-under stating the computing resources assigned to a specification.	Mod
9. Conflicting Inter-system Specification	Requirements of cooperating systems, or parent/embedded systems, which taken pairwise are incompatible.	System
10. Incorrect or missing external constants	Specification of an incorrect value or variable, or a missing value or variable in a requirement.	Mod
11. Incorrect or missing description of initial system state	Failure to specify the initial system state, when that state is not equal to 0.	Mod

Type	Description	Occurs
12. Overspecification of Requirements	Requirements or specification limits that are excessive for the operational need, causing additional system cost.	System, Sub
13. Incorrect input or output descriptions	Failure to fully describe system input or output.	Mod

Table 3-2. Expanded Requirement Fault Taxonomy.

Type	Description	Occurs	Source	Development phase where it is earliest detectable
.0 Requirements	Originate in Requirements phase; found in the Requirements Specification			
.1 Incomplete Decomposition	Failure to adequately decompose a more abstract specification.	System, Sub, Mod	[2], [13], [14]	Requirements
.2 Omitted Requirement	Failure to specify one or more of the next lower levels of abstraction of a higher level specified.	System, Sub, Mod	[2], [4], [9], [10], [11], [13]	Requirements
.3 Improper Translation	Failure to carry detailed requirement through decomposition process, resulting in ambiguity in the specification.	System, Sub, Mod	[2], [3], [4], [8], [9], [10], [11], [13], [14]	Requirements
.4 Operational Environment Incompatibility	Specification which does not accommodate the operational environment, such as data rates, data formats, etc.	System, Sub, Mod	[2], [3], [8], [13]	Requirements
.5 Incomplete Requirement Description	Failure to fully describe all requirements of a function.	Mod	[2], [3], [10], [11], [13], [14]	Requirements
.6 Infeasible Requirement	Requirement, which is unfeasible or impossible to achieve given other system factors, e.g., process speed, memory available.	Mod	[2], [13]	Requirements
.7 Conflicting Requirement	Requirements that are pairwise incompatible.	System, Sub, Mod	[2], [3], [4], [8], [9], [10], [11], [13], [14]	Requirements
.8 Incorrect Assignment of Resources	Over-or-under stating the computing resources assigned to a specification.	Mod	[2], [10], [11], [13]	Requirements
.9 Conflicting Inter-system Specification	Requirements of cooperating systems, or parent/embedded systems, which taken pairwise are incompatible.	System	[2], [3], [10], [11], [13], [14]	Requirements
.10 Incorrect or missing external constants	Specification of an incorrect value or variable, or a missing value or variable in a requirement.	Mod	[2], [10], [11], [13]	Requirements
.11 Incorrect or missing description of initial system state	Failure to specify the initial system state, when that state is not equal to 0.	Mod	[2], [10], [11], [13]	Requirements

Type	Description	Occurs	Source	Development phase where it is earliest detectable
.12 Overspecification of Requirements	Requirements or specification limits that are excessive for the operational need, causing additional system cost.	System, Sub	[2], [4], [7], [9], [13]	Requirements
.13 Incorrect input or output descriptions	Failure to fully describe system input or output.	Mod	[2], [10], [11], [13]	Requirements
.14 Not traceable	Requirement which is specified but not implemented. Items cannot be traced to the appropriate previous or subsequent phases.	System, Sub, Mod	[10], [11], [14]	Design
.15 Unachievable item	Requirement that is specified but difficult to achieve. The requirement statement or functional description cannot be true in the reasonable lifetime of the product	System, Sub, Mod	[10], [11]	Design, coding
.16 Non-verifiable Item	The Requirement statement or functional description cannot be verified by any reasonable testing methods	System, Sub, Mod	[10], [11], [14]	Design, coding, testing
.17 Wrong Section	Information which is in a different section in requirements document	System, Sub, Mod	[13]	Requirements
.18 Intentional Deviation from specifications	The Requirement which is specified at higher level but intentionally deviated at lower level	System, Sub, Mod	[13]	Requirements

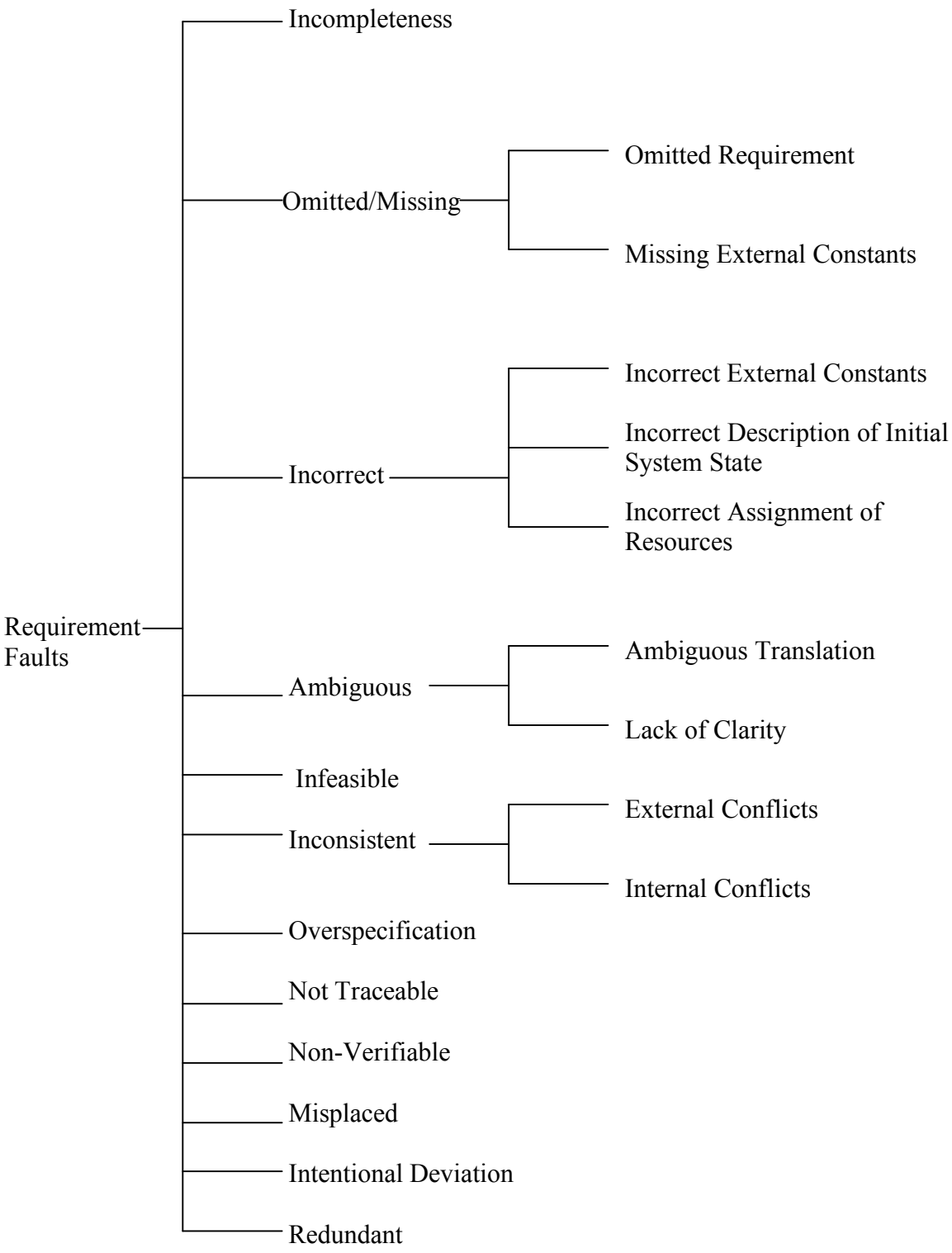


Figure 3-1. Connectors of Requirement Faults.

Table 3-3. New Revised Requirement Fault Taxonomy.

Major Fault	Sub-Faults	Description of Sub-Faults	Original taxonomy faults mapped here
.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 pairwise incompatible. .6.2 Requirements of cooperating systems, or parent/embedded systems, which taken pairwise are incompatible.	.7,.9
.7 Overspecification	----------	.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

Major Fault	Sub-Faults	Description of Sub-Faults	Original taxonomy faults mapped here
.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	----------

As our three independent analysts performed the activities and implemented the processes associated with Tasks 4 and 7 (Sections 3.4 and 3.7 respectively), changes and additions were made to the revised taxonomy generated by Task 1 (Table 3-3 above). The final generic taxonomy clarified and generated for NASA is discussed under Task 7 (see Table 3-11).

3.2 Task 2 - Presentation and Milestone Meeting 2 (PMR2)

Presentation materials were developed and presented at Milestone Meeting 2 (PMR2) on 3/29/02. PMR2 covered progress to date and outlined issues faced by the project. The presentation is not included as part of this final report but can be provided again upon request.

3.3 Task 3 - Presentation and Milestone Meeting 3 (PMR3)

Presentation materials were developed and presented at Milestone Meeting 3 (PMR3) on 6/28/02. PMR3 covered progress to date and outlined issues faced by the project. The presentation is not included as part of this final report but can be provided again upon request.

3.4 Task 4 - Examine NASA-Specific Requirements Faults

Obtaining NASA project-specific fault data has proven to be difficult and it was noted that the level of fault data detail provided varied greatly. However, we did receive and examine IV&V "comments" on requirement problems for four projects and Project fault reports (requirements) for another two projects. The data received was very useful. Analysis of the data resulted in changes to our generic fault taxonomy and to our taxonomy extension/tailoring processes.

Four projects provided data. Data from two projects was in the form of Project Fault Reports. The data for the other projects was in the form of IV&V "comments" on requirement problems.

Three analysts independently examined and categorized project faults for the six data sources. Each analyst followed the fault taxonomy extension process for NASA software classes. The fault taxonomy extension process, broken down into Process A and B, was developed for Task 6 (see Section 3.6). In implementing Process A (see Section 3.6.3.3), they each started with the new revised generic taxonomy that was produced as the final product for Task 1 (see Table 3-3). During this process 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, discussed in Section 4.2, Recommendations, would be for the projects and IV&V reviewing analysts to document each individual fault separately.

Further, in consultation with the primary researcher, the analysts also shared lessons learned during the process, as they encountered them, to ensure that in the final analysis, categorization metrics collected from each analyst could be directly compared (i.e., apples and apples). Some of the lessons learned resulted in revisions to Process A. These lessons also resulted in adding new generic taxonomy categories, clarifying existing category definitions, and shedding new light on the orthogonality of the taxonomy.

Section 3.7, Task 7, discusses the clarifications and additions made to the revised taxonomy based on lessons learned. Table 3-11 in Section 3.7 displays the final generic taxonomy categories developed for NASA.

3.5 Task 5 - Build a List of IV&V Techniques

Table 3-4 below is a full list of IV&V activities that are commenced and/or completed during the Software Requirements Phase of NASA Code-S Missions. For any particular Code-S project, based on a criticality/CCHR assessment, only a relevant subset of these activities are recommended, project-approved, and then implemented. The activities

described in the table below are not listed in any specific order. The table includes the activity/technique, a description, and purpose.

The information was gathered from the following sources: NASA Code-S Statement of Work (SOW), NASA IV&V website, SAIC NASA IV&V Methods document (for SAIC Risk Cube), IV&V Activities PowerPoint presentations to a NASA project, SAIC NASA IV&V Formulation Phase and Project Plan Documents/Templates, SAIC Kernels, other internal SAIC IV&V documents/artifacts (e.g., SAIC IV&V Division Management Guide/Plan), and industry experience of an IV&V software engineer contributing to this project.

Table 3-4. NASA IV&V Techniques.

Activity/Technique	Description	Purpose
Documentation Reviews	Critically evaluate system documentation based on inspection criteria tailored to the document's purpose and maturity. Evaluation criteria defined by NASA-STD-2100-91: • The documentation goals of the project are adequately satisfied. • Clear descriptions of the software management, engineering, and assurance processes and products are provided. • Consistency of format across the project documentation is achieved. • Traceability to the untailored Standard is maintained. • Traceability between products of each phase of the development life cycle is maintained.	Verify that the following core set of documents exist, are adequate for the purpose of managing project software development activities, and are current: • Software/Product Development Plan • Configuration Management Plan(s) • Quality Assurance Plans(s) • Software Systems Specification(s) • Integrated/Individual SW Development Schedule(s) • Software Test Plan(s)
Technical Reviews & Audits—Process Audits	Independently review program process data and information for the purpose of assessing compliance of actual practices with established processes.	To determine the effectiveness of process implementation. To detect and report risk areas based on discrepancies between established processes and everyday practices. To identify areas of process improvement
Software Requirements Analysis--Modeling with Tool	Examine tools and methods used by developer to model sophisticated requirements.	To verify software requirements are correctly modeling using appropriate tools.

Activity/Technique	Description	Purpose
Traceability Analysis—Requirements	Trace various levels of requirements throughout the software development lifecycle. Trace system requirements to software requirements, software requirements to design, design to code, code to test. Perform a trace both forwards and backwards, meaning all high-level requirements are satisfied by lower-level requirements and all lower-level requirements are derived from high-level requirements.	To verify the decomposition of system and the software throughout the development lifecycle; identify requirements not implemented; determine test program coverage.
Software Requirements Analysis	Evaluate requirements documentation based on a predefined set of criteria tailored to the documentation's purpose and maturity. Examples of criteria include: correctness, consistency, completeness, performance, reliability, constraints, organization, compliance, accuracy, readability, and testability.	To assess how well the requirements documentation satisfies software system objectives; To ensure an accurate translation between higher and lower level requirements documents
Interface Requirements Analysis	Investigate issues, questions, comments, violations, discrepancies, or deviations between interface requirements and their related software requirements for CCHR related areas.	To verify software requirements are correctly reflected in interface requirements documents; To ensure that assumptions, which are implied in the requirements documents, are consistent with specific requirements in those same documents.
Inspection (Requirements)	Verify that the requirements meet customer needs, can be implemented, and are complete, traceable, testable, and consistent so that omissions, defects, and ambiguities in the requirements are detected. The inspection process may consist of multiple steps for the segregation of the inspection functions of: 1) Inspection planning; 2) Product overview; 3) Inspection preparation; 4) Examination meeting; 5) Defect rework; 6) Resolution follow-up.	To find, classify, report and analyze defects in the product. To detect anomalies and problems and verify their resolution by the author. An inspection is performed by a small team of peer developers and includes, but is not led by, the author. The inspection team usually consists of three to six persons, and in some cases includes personnel from the test group, quality assurance, or V&V. The participants assume specific roles in order to find, classify, report and analyze defects in the product. Each type of inspection is specifically defined by its intended purpose, required entry criteria, defect classification, checklists, exit criteria, designated participants, and its preparation and examination procedures. Inspections do not debate engineering judgments, suggest corrections, or educate project members; they detect anomalies and problems and verify their resolution by the author.
Walkthroughs (Requirements)	Participate in a walkthrough of the requirements specification to ensure that the software requirements are correct, unambiguous, complete, verifiable, consistent, modifiable, traceable, testable, and usable throughout the life cycle.	To participate in the evaluation processes in which development personnel lead others through a structured examination of a product. To ensure that the participants are qualified to examine the products and are not subject to undue influence.

Activity/Technique	Description	Purpose
Formal Requirements Review	Participate in formally evaluating the adequacy of project/system/software requirements with customer representatives typically during the requirements stage of the development lifecycle (e.g., at SRR and possibly PDR). Support the customer review by reviewing available documentation and performing other tasks requested by the project (for example, creating a review checklist).	To provide insight to the status of risk items associated with requirements in their current state; identify action items; aid in determining the overall condition of the project/system/software requirements with respect to technical issues and schedule; assist in improving communication between project members.
Special Studies—Operational Concept Analysis	Analyze the manner in which the software system interacts with and is dependent upon states of the environment and external decisions, especially of human operators. Use available project artifacts, information and software demonstrations, if possible.	To make certain the implemented software meets intended operational concepts.
Software Design Analysis—General Review	Review the technical adequacy of the design according to detailed pre-established set of criteria and procedures.	To ensure that the software/system design meets requirements; aid in evaluating and mitigating risk associated with the design; assist in determining if the project is ready to advance to the next stage in the development lifecycle.
IV&V Analysis of V&V Test Program	Perform analyses of the developer's testing program to ensure complete and adequate test coverage and specification.	To verify the program limits are correctly stated and implemented.
Evaluation of Software Test Environment/Facilities	Evaluate the test environment for suitable tools, simulations, hardware and software.	To verify that the test environment and facilities are sufficient for verifying system requirements and as-built functionality.
Reusability Assessment	Assess the use of commercial-off-the-shelf (COTS) software, modification of existing software, and the use of code modules specifically designed for reuse.	Two important tasks to verify are: 1) to identify dependencies on the original hardware or software operating environment; 2) to verify that the human interface will function correctly in the new target environment. Reuse of existing software can cost-effectively improve the quality of a software product (and reduce requirement scope creep and reduce injection of requirement related defects).
Security Assessment	Evaluate the security controls (requirements) on the system to ensure that they protect the hardware and software components from unauthorized use, modifications, and disclosures, and to verify the accountability of the authorized users.	To verify that these controls (requirements) are appropriate for achieving the system's security objectives. A system security assessment will include both the physical components (e.g., computers, controllers, networks, modems, radio frequency, infrared devices) and logical components (e.g., operating systems, utilities, application programs, communication protocols, data, administrative operating policies and procedures).

Activity/Technique	Description	Purpose
Develop the Catastrophic/Critical/High Risk (CCHR) Functions List (CFL)	Develop, maintain, and deliver a Catastrophic/Critical/High Risk (CCHR) Functions List (CFL) as the basis for planning IV&V areas of concentration and work prioritization.	To develop a report that lists all of the functions which involve software that are classified as having catastrophic or critical safety risk, or have high technical or developmental risk. For each item, the rationale for list inclusion is identified, as well as all software involved in the performance of the function at the Computer Software Configuration Item – CSCI (and Computer Software Component - CSC) level if necessary to define IV&V efforts clearly. How each software entity is involved in the function is identified. The listing also includes SAIC's recommendations on which functions should not be IV&V'd or receive reduced or enhanced analyses, the rationale for each recommendation, and the NASA IV&V Project manager's final decision on those recommendations
Issues Tracking	Report on a monthly basis all significant issues involving CCHR software. Issues may cover any part of the software development lifecycle for CCHR software (e.g., requirements, design, code, test, etc)	To keep NASA Management informed about the status of all issues affecting CCHR software. Part of required monthly reporting to NASA. Can be included as one of the sections of the Task Order Monthly Progress Report.
Metrics Assessment	Utilize project software development metrics, and knowledge gained through other sources (e.g., analyses efforts) to assess project ability to comply with project requirements and schedules.	To identify and report on deficiencies throughout the life-cycle and provide the results as part of the Monthly Software Status Report. The metrics to be assessed include, but are not limited to: Processor sizing; Processor timing; Mass Memory sizing; Software Development, Test, and Integration Progress; and Software Errors.
Change Impact Analysis	Technically assess all proposed changes (e.g., Engineering Change Proposals, and Discrepancy Reports) that are associated with selected CCHR function areas or affect processes associated with those areas to evaluate the impact on those function areas. Assessments may be required for changes to flight rules, operational procedures, hardware, software, and system requirements to assess the impact on project software receiving IV&V.	To assess and determine if the changes are complete, meet the intent of the change, are necessary, and ensure that all performance and operational usage impacts are identified. Assessment results are reported in the format of an approved Analysis Report.
IV&V Test Planning	For selected CCHR software functions, recommend independent testing with the objective of verifying agreement between software and software specifications and demonstrating the software's adequacy to perform the mission.	To complement rather than duplicate the project software developer's testing. The recommendation for independent testing is to be submitted to the NASA IV&V Project Manager at least 90 days prior to any planned IV&V testing as an IV&V Test Plan. The test plan includes the objectives, scope, program value, and required resources of the test being proposed. Prior to its execution the NASA IV&V Project Manager approves proposed IV&V testing.

Activity/Technique	Description	Purpose
IV&V Project Planning	Define the recommended level of IV&V support for the project. Recommendation is founded upon a software risk criticality assessment conducted during the IV&V Formulation Phase.	To recommend a set of IV&V risk–reduction activities tailored specifically to the Project's software development and to list the IV&V activities per CCHR Function List (CFL) item. To provide a schedule for executing the recommended implementation phase activities, correlated with the project's schedule and to list the IV&V deliverables and due dates based on the IV&V Statement of Work (SOW).

3.6 Task 6 - Adopt or Build a Method for Extending the Taxonomy

We have built and adopted a method for extending the taxonomy as described in the next few subsections. Subsection 3.6.1 provides an introduction to Task 6 explaining why it is crucial to detect faults at the requirements phase of the software development life cycle and our overall process for extending a fault taxonomy to avoid faults at the requirements stage. Subsection 3.6.2 lists standard terms and definitions used throughout this section. Subsection 3.6.3 is broken down into the following subsections: Motivation, NASA Software Classes, Process (A) for Extending a Fault Taxonomy for a Project Class, and Process (B) for Extending a Class Taxonomy for a Project. Under the motivation subsection, we discuss the challenges in defining a process to extend a fault taxonomy and the inspiration for our taxonomy. We also discuss our high level process of extending a fault taxonomy with the help of a figure. The NASA software classes subsection discusses the criteria for the classification of NASA projects. We split our process for extending a fault taxonomy into two parts: Process A and Process B. Process A discusses all the activities that are to be performed to develop a class-specific taxonomy. The outputs of Process A are inputs to Process B (i.e., we take a class-specific taxonomy and perform all the activities described in the Process B section to develop a project-specific taxonomy). Subsection 3.6.4 provides a summary of our accomplishments. Throughout this section, many tables and figures have been provided to enable a clear and better understanding of our process to extend a fault taxonomy.

3.6.1 Introduction

Understanding faults at the requirements stage saves effort, time, and cost and helps developers build correct and reliable software. Early detection and correction of faults at the requirements phase is less expensive than detecting faults during the design, coding, and/or testing phase [7]. Task 6 focuses on faults at the requirements stage of the software life cycle to improve Independent Verification and Validation (IV&V) and also to reduce the cost, time, and effort of eliminating faults at advanced stages of the software life cycle.

Discussed within this section is a process to extend a fault taxonomy. We split our effort into two processes: Process A and Process B. For Process A, we take our generic fault taxonomy and NASA project requirement faults/problem reports and then perform the activities of categorizing the faults according to the generic fault taxonomy. We then determine fault types for NASA software classes. The outcome of these activities is a

class-specific taxonomy. For Process B, we begin by taking a class-specific taxonomy, requirement faults and/or problem reports for projects and project specific goals/priorities. We then perform the activities associated with categorizing the faults for a project according to a class-specific taxonomy. The outcome of this process is a project-specific taxonomy.

In this section, we have also considered some optional activities that will be useful for NASA projects. The optional activities include estimating the effect of the requirement faults on a system and the probability of its occurrence. The critical requirement faults can also be identified for a project under a class.

3.6.2 Definitions

Refer to Section 2, Standard Definitions, for definitions relevant to the taxonomy process as described for Task 6.

3.6.3 Process to Extend a Fault Taxonomy

3.6.3.1 Motivation

Challenges in taxonomy creation and materials useful to building a taxonomy have been identified. Originally it seemed that taxonomy creation was straightforward. However, as the research effort progressed, we became aware of complexities.

We searched for papers for a method to extend the fault taxonomy from a generic taxonomy to a class-specific taxonomy. There are no papers for extending a fault taxonomy in the literature. We also searched for papers in areas besides software engineering, including websites such as the Digital Library Network for Engineering and Technology (DLNET). We also examined resources such as IEEE, ACM Digital Library, Altavista, Hotbot, Google, Yahoo, WilsonWeb, Kluwer, CSA, Cite seer search engines, CERT Coordination Center (SEI), NASA website, and NASA IV &V Facility online resources and libraries.

With a lack of an existing method, we defined a new process to extend a fault taxonomy. The inspiration for this taxonomy was a paper entitled “Towards a Taxonomy of Software Connectors” [2]. The paper presents a comprehensive classification framework and taxonomy of software connectors. Connectors manifest themselves in a software system as shared variable accesses, table entries, buffers, instructions to a linker, procedure calls, networking protocols, SQL links between a database and an application, and so forth. In large and especially distributed systems, connectors become key determinants of system properties, such as performance, resource utilization, global rates of flow, scalability, reliability, security, evolvability, etc. The classification supports a deeper understanding of existing connectors and their relationships. It also provides the information needed to design new, more powerful connectors by combining existing mechanisms [2].

The taxonomy is obtained through an extensive analysis of existing component interactions, as software systems are composed from prefabricated, heterogeneous components that provide complex functionality and engage in complex interactions. The

paper demonstrates the use of the taxonomy on the architecture of a large, existing system. The overall structure of the connector classification framework can be viewed in Appendix A. Each connector is identified by its primary service category and further refined based on the choices made to realize these services [2]. For example, for our taxonomy we start at the main system (e.g., manned flight) and keep adding connectors. The next level of connectors we add are major requirement faults, and then to sub-faults of requirements faults.

Our process for extending a fault taxonomy is shown in Figure 3-2. The process builds on our generic taxonomy, which is a major enhancement of the fault taxonomy worked on in [4] and is discussed in this Final Report. First, we take our generic fault taxonomy, NASA project requirement faults and problem reports and perform Process A as discussed in Section 3.6.3.3. The result is a taxonomy for a NASA software project class. The criteria for the classes are shown in Section 0. Based on Project Task 1, 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, we perform Process B as discussed in Section 3.6.3.4. 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. Tolerance analysis is discussed in Section 3.6.3.4.

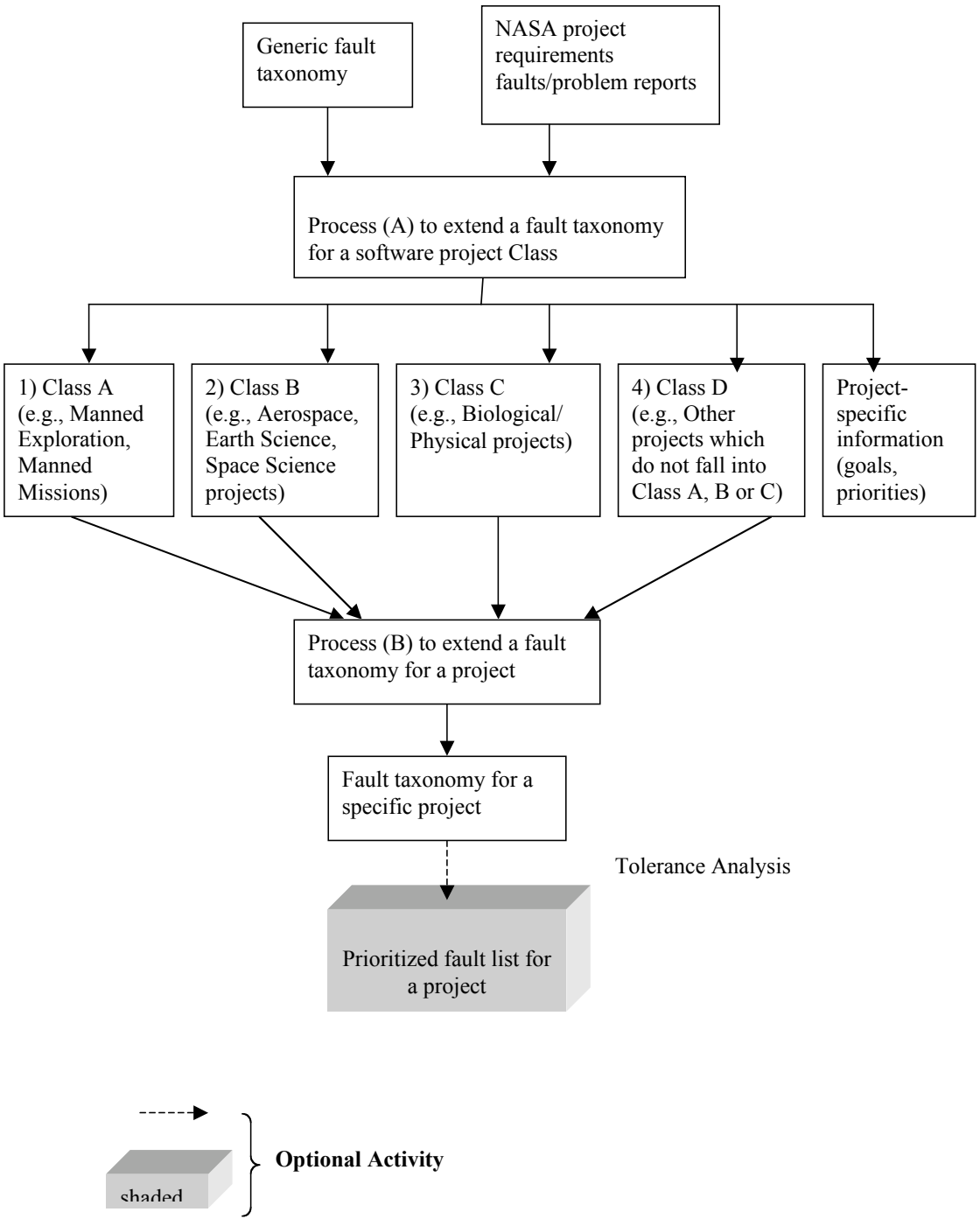


Figure 3-2. High Level Process to Extend a Fault Taxonomy.

3.6.3.2 NASA Software Classes

The software classification scheme is incorporated from the NASA software safety standard [3]. The result is a four-class structure tagged Class 'A', 'B', 'C', and 'D'. A software category is determined according to the following procedure [3]:

1. Begin at 'Class A'. If any criteria are met, then software is 'Class A'; else continue
2. At 'Class B', if any criteria are met, then software is 'Class B'; else continue
3. At 'Class C', if any criteria are met, then software is 'Class C'; else software is 'Class D' (i.e., none of the Class A, B, or C criteria are met).

The software classification is made according to the following criteria:

(1) "Class A" when any of the following conditions are met:

a) Potential for loss of life – Yes

b) Potential for loss of equipment – Greater than $100M

c) Potential for waste of resource investment – Greater than 200 work-years on software

d) Software control category IA (from NASA Software Safety Guidebook [5])

(2) "Class B" when any of the following conditions are met:

a) Potential for serious injury – Yes

b) Potential for catastrophic mission failure – Yes

c) Potential for loss of equipment – Greater than $20M

d) Potential for waste of resource investment – Greater than 100 work-years on software

e) Software control categories IIA and IIB (from NASA Software Safety Guidebook [5])

(3) "Class C" when any of the following conditions are met:

a) Potential for partial mission failure – Yes

b) Potential for loss of equipment – Greater than $2M

c) Potential for waste of resource investment – Greater than 20 work-years on software

d) Software control categories IIIA and IIIB (from NASA Software Safety Guidebook [5])

(4) "Class D" - Software not meeting any of the above criteria. For example:

a) Potential for loss of life or serious injury – No

b) Potential for loss of equipment – less than $2M

c) Potential for waste of resource investment – less than 20 work-years on software

d) Safety control category IV (from NASA Software Safety Guidebook [5])

It should be noted that numerous discussions transpired among NASA civil servants and the researchers to determine the class of a project. Much of the criteria (such as dollar amount for equipment) is difficult to obtain and/or subjective. The classes assigned in Task 4 and 7 were finally reached by consensus using available information.

3.6.3.3 Process (A) for Extending a Fault Taxonomy for a Project Class

The Process for developing a class-specific requirement fault taxonomy is shown in Table 3-5. The table consists of six fields: entry criteria, activities, exit criteria, inputs, outputs, and process controls and metrics. 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.

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 (as shown in Table 3-6), 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 (as depicted in Table 3-7).

We will use Table 3-6 to estimate fault frequency for different projects under each class. For example, we will use the table 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. As shown in the example in Table 3-6, 50 incomplete decomposition requirement faults exist in the Class B project and 10 incomplete description faults exist. Overall 45 requirement faults were found for Class B. The percentage of occurrence of incomplete requirement faults is 23% for Class B.

Finally, we will determine the historically most probable requirement faults for each class as shown in Table 3-7. As shown in the example in Table 3-7, for manned exploration and

manned mission projects under Class A the most critical sub-faults are incomplete decomposition and incomplete requirement description under major fault incomplete requirement fault (plus other sub-faults). We list the top five major and sub-requirement faults for Class A projects (as shown in the example in Table 3-7). We will 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 all versions of our requirement fault taxonomy are properly maintained under configuration control. Also NASA project data must be maintained by class. Process metrics include person hours for the effort, number of projects, number of requirements faults, etc.

Table 3-5 . Process (A) 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

Table 3-6. Estimation of Fault Frequency for Software Requirement Fault Types.

S/W Requirement Fault Types	Count of Fault Frequency	% of Fault Occurrences
1) Major Fault: Incomplete 0.1 Incomplete Decomposition 0.2 Incomplete Requirement Description	 9 1	 20 % 2 %
: :		
N) New Fault 0.n Subfault		
Totals	45	100%

Table 3-7. Determination of Critical Requirement Faults for a System.

System	Historical Top 5 Most Probable Function Areas (Critical Requirement Faults)
Class A (e.g., Manned Exploration, Manned Mission)	1): Incomplete .1: Incomplete Decomposition .2: Incomplete Requirement Description 2): 3): 4): 5):
Class B (e.g., Aerospace, Earth Science, Space Science)	1): 2):
Class C (e.g., Biological/Physical)	1): 2):
Class D (e.g. projects that do not fall under Class A, B or C)	1): 2):

3.6.3.4 Process (B) for Extending a Class Taxonomy for a Project

Similarly, the process involved in developing a project-specific requirement fault taxonomy is shown in Table 3-8. The table consists of six fields: entry criteria, activities, exit criteria, inputs, outputs, and process controls/metrics. The entry criteria field describes a checklist of pre-conditions that must be met before process activities can start. All the information and data needed such as a class-specific fault taxonomy, requirement faults and/or problem reports for projects, and project specific goals/priorities 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.

The activities to be performed include selecting a class-specific requirement fault taxonomy, and examining problem reports or requirement faults of projects. We perform an additional optional activity of tolerance analysis for each project as follows. From the

fault types identified in Table 3-6, we determine a tolerance factor for each requirement fault to corresponding function areas (e.g., flight safety function) for each project (e.g., manned flight) as shown in Table 3-9. We assign a tolerance factor on a scale of 1 to 10. If the tolerance factor for a fault type is 10 then it has a potential severe effect on the system and a tolerance factor of 1 has no effect on the system. For example, the tolerance factor for requirement fault lack of clarity for flight safety and manual safety function is severe for manned systems that fall under Class A and is therefore assigned a tolerance factor of 10. On the other hand, for a communication function it is 7. This indicates that compared to the previous fault it might not be as severe but still might have significant effect on the system as shown in Table 3-9.

Next, as shown in Table 3-10, the requirement fault and corresponding feature of the system are combined and both the probability of a faults' historical occurrence and fault exposure for the system is determined. As shown in the example in the first row in Table 3-10, the lack of clarity to flight safety feature pair has a tolerance factor of 10 which is severe and its historical probability of occurrence is also high at 0.9. Finally, as shown in Table 3-10, we calculate fault exposure. Fault exposure is the product of the tolerance factor and the probability of its occurrence. It is similar to risk exposure [6]. From these calculations, the most critical aspects are identified. For example, the * symbol in the first row of Table 3-10 with 9.0 fault exposure value indicates that this fault occurring for this function area could have a severe effect on the system.

The outputs of this process are the frequency counts of the faults and the crucial requirement fault categories for the project. The optional output, if we perform the optional activity, is a prioritized fault list for the project. From the fault exposure values, we prioritize a list of faults as critical (that could have critical effect on the system). The fault exposure values with severe effect are indicated by the * symbol for requirement fault to function area pairs. We repeat this process for the project until our exit criteria is met (i.e., we have developed a project-specific requirement fault taxonomy). Similar to Process A, the process controls ensure that all versions of our requirement fault taxonomy are properly maintained under configuration control. Also, NASA project data must be maintained by class. Process metrics include person hours for the effort, number of projects, number of requirement faults, tolerance factors, historical probability of occurrence, and fault exposure values, etc**.**

Table 3-8. Process for a Project-Specific Fault 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 Class-specific requirement fault taxonomy 2. Examine NASA specific projects in selected class 3. Categorize the faults for the project according to the class-specific fault taxonomy 4. Determine the frequency of faults for the project	1. A project-specific requirement fault taxonomy has been developed

Entry Criteria	Activities	Exit Criteria
	5. Identify the crucial fault categories for the project **Optional Activity:** 6. Estimate the tolerance of each function area of a project to a corresponding requirement fault (Table 3-9) 7. Determine the historical probability of occurrence of the fault (Table 3-10) 8. Calculate the product of probability of occurrence and tolerance factor to determine the critical requirement faults (Table 3-10)	
Inputs	**Process Controls/Metrics**	**Outputs**
1. Class-specific fault taxonomy 2. Requirement faults/problem reports for the project 3. Project specific information (goals, priorities)	Process Controls: 1. Maintenance of configuration control of taxonomy 2. Maintenance and management of NASA project data by Class Metrics: 1. Person Hours for effort 2. # projects 3. # faults 4. Tolerance Factors 5. Historical Probability of Occurrence 6. Fault Exposure Values	1. Frequency counts of faults 2. Crucial fault categories for the project 3. Prioritized fault list for the project (**Optional activity**)

Table 3-9. Estimation of Tolerance Factor for a System Function Area to Requirement Fault Type.

System Type and Class	Function Areas / Critical Requirement Fault	**Flight Safety**	**Manual Safety**	**Communication**									
Manned, Class A	Lack of Clarity	10	10	7									
	Inconsistent	7	7	2									

Table 3-10. Calculation of Fault Exposure.

Critical Requirement Fault to Function area (row and column combined from Table 3-9)	Tolerance	Historical Data on Probability of Occurrence	Fault Exposure= Tolerance*probability
Lack of clarity/flight safety	10	0.9	9.0 *
Lack of clarity/manual safety	10	0.9	9.0 *
Inconsistent/flight safety	7	0.7	4.9
Inconsistent/manual safety	7	0.8	5.6
	1	1.0	1.0
	8	1.0	8 .0 *

* = severe effect on system

3.6.4 Summary

In summary for Task 6, we have built a process to extend a requirement fault taxonomy. We use two processes, Process A and Process B, to develop a class-specific taxonomy and a project-specific taxonomy. Our method helps in identifying the most probable critical requirement faults for a system. We defined the additional optional activity of estimating the tolerance effect of a requirement fault on a system, in order to seek improvement in that area.

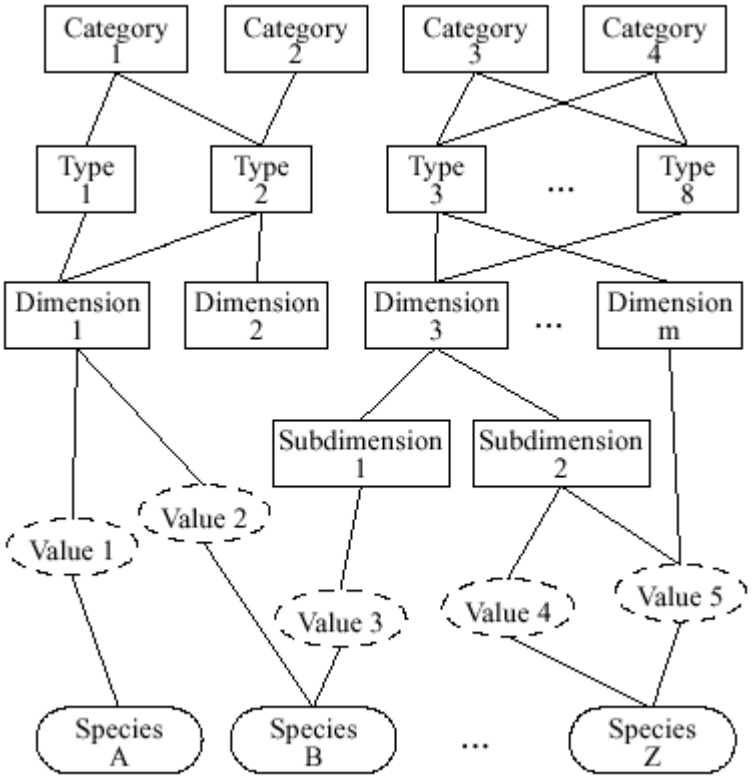


Figure 3-3. Structure of the Connector Classification Framework.

3.7 Task 7 - Implement the Method to Extend the Fault Taxonomy

This task entailed implementing the method to extend the fault taxonomy to develop a class-specific taxonomy (Process A) and a project-specific taxonomy (Process B). Prior to beginning Process A, 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. These NASA classes are described in Section 3.6.3.2.

Our three analysts followed all steps for Process A starting with the revised generic taxonomy resulting from Task 1. During this process, lessons learned from each analyst resulted in revisions, clarifications, deletions, and additions to that generic taxonomy. Insight was also gained during a review of the orthogonality concept as applied to these taxonomy categories. Our conclusions from lessons learned on orthogonality are discussed in Section 4.1. The final generic requirement fault taxonomy for NASA is represented in Table 3-11 below.

The three analysts in conjunction with the primary researcher made the following changes to the revised generic taxonomy (see Table 3-3) from Task 1:

a) Descriptions of the several Fault and Subfault categories were clarified to reduce confusion among present and future NASA 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 Table 3-11. All of the fault category item descriptions were clarified except for Category 7.

b) 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 and 1.2 were combined; Subcategories 4.1 and 4.2 were combined; and Categories 5 and 9 were combined. Category 9 is now "Reserved for future."

c) The following subcategory was deleted/removed from the taxonomy, again due to orthogonality or similarity issues: Subcategory 2.3.