A Case Study of Measuring Process Risk for Early Insights into Software Safety

Lucas Layman
Fraunhofer CESE
College Park, MD
llayman@fc-md.umd.edu

Victor R. Basili
Marvin V. Zelkowitz
U. of Maryland and Fraunhofer CESE
College Park, MD
{basili,mvz}@cs.umd.edu

Karen L. Fisher
NASA Goddard Spaceflight Center
Greenbelt, MD
karen.l.fisher@nasa.gov

ABSTRACT

In this case study, we examine software safety risk in three flight hardware systems in NASA's Constellation spaceflight program. We applied our Technical and Process Risk Measurement (TPRM) methodology to the Constellation hazard analysis process to quantify the technical and process risks involving software safety in the early design phase of these projects. We analyzed 154 hazard reports and collected metrics to measure the prevalence of software in hazards and the specificity of descriptions of software causes of hazardous conditions. We found that 49-70% of 154 hazardous conditions could be caused by software or software was involved in the prevention of the hazardous condition. We also found that 12-17% of the 2013 hazard causes involved software, and that 23-29% of all causes had a software control. The application of the TRPM methodology identified process risks in the application of the hazard analysis process itself that may lead to software safety risk.

Categories and Subject Descriptors
D.2.8 Process Metrics;

General Terms
Management, Measurement, Verification.

Keywords

Constellation program; hazard reports; measurement; safety; empirical software engineering

1. INTRODUCTION

Development of large complex systems in the aerospace, defense, energy and transportation industries requires constant attention to safety risks. A *safety risk* is a risk whose effect can be injury or the loss of life either directly or through a chain of events. Software safety risk has become a greater concern in systems development as many traditionally hardware-centric systems become more reliant on software.

In this paper, we present a case study measuring software safety risk in the multi-year, multi-billion dollar Constellation spaceflight program at NASA. Software safety risk is a form of *technical risk*, where flaws in the design and implementation can lead to system failure, loss of the mission or loss of life. Quantifying technical risk in software often relies on testing and simulation, which require either working code or a concrete design. In the Constellation program, however, program and project managers need insight into the state of software safety in order to guide the design process. The challenge becomes: *how can we gain early insight into the state of software safety?*

To discover and prevent technical risks, Constellation uses a number of established Reliability, Safety and Mission assurance (RSMA) processes for managing development, uncovering risks, and mitigating their effects, such as peer review, fault-tree analysis, testing, failure-modes and effects analysis, and more. In the context of software safety, RSMA processes are meant to create a more robust and fault-tolerant system design and verify the correct implementation of the design. These RSMA processes, however, can be the source of *process risk*, which emerge when the processes are not performed appropriately, are not appropriate for the situation and/or they are not well-defined. Process risk leads to technical risks, which has been demonstrated by a number of notorious software safety failures that are at least partly attributable to the development process [6,7].

In this case study, we apply the six-step Technical and Process Risk Measurement (TPRM) methodology that leverages the relationship between process and technical risk to gain early insight into software safety risk on the Constellation program. We apply the TPRM methodology using the artifacts of one RSMA process, hazard analysis, to provide NASA quality assurance managers with metrics on the state of software safety risk during the early design phases of the Constellation program. The application of the TPRM methodology also uncovered a number of process risks in the definition and application of the hazard analysis process.

The remainder of this paper is organized as follows: Section 2 provides an overview of the TPRM methodology; Section 3 describes the development context of the Constellation program; Section 4 describes the application of the TPRM methodology to create measures of software safety risk and the technical and process risks uncovered in this study; and Section 5 concludes with a discussion and future work.

2. THE TECHNICAL AND PROCESS RISK MEASUREMENT METHODOLOGY

The purpose of RSMA processes is to improve safety, reliability and mission assurance in a product. Thus, we assume that RSMA process artifacts should contain information pertaining to the

ICSE'11, May 21-28, 2011, Honolulu, HI, U.S.A.
Copyright © 2011 ACM 978-1-4503-0445-0/11/05... $10.00

ongoing state of reliability, safety, and mission assurance in the product while it is being developed. The process artifacts form the basis of our measurement. By leveraging processes and process artifacts, the approach provides early insight throughout the development process because it does not depend solely on the incremental or final product. The purpose of the TPRM methodology is to assist organizations in measuring technical and process risk throughout development.

The TPRM methodology [3] starts at a high-level to investigate risk measurement possibilities and then is iteratively tailored down to develop specific measurements and responses to specific risks. The methodology is composed of the following six-steps:

1 Identify **insight areas** from the RSMA process that provide insight into risk areas. What process artifacts can potentially be sources of risk information?

   Examples: test results, design documents, hazard reports

2 Identify the **measurement opportunities** that provide insight into each risk area. What information contained in the process artifacts is useful for identifying areas of risk?

   Example: lists of single points of failure in design documents

3 Develop **readiness assessment questions** to provide a quick status of the risk and to identify if it is possible to delve deeper into the area. Is sufficient information being collected to measure potential risk using this artifact?

   Examples: Is the "software component" column of the test result report always filled out? Are the design documents detailed enough to get an adequate picture of the critical software components?

4 Define **goals and questions** for each risk area to expose risks associated with RSMA process artifacts. What specific risk questions do we want to answer using the insight areas?

   Example: Have we mitigated all hazard causes? Have we fixed all severe criticality bugs discovered during test?

5 Develop and enumerate **measures and models** of how the metrics will be **interpreted** via threshold values. What are specific measures and what do they mean?

   Example: Count the number of test failures identified for Component X; one or more test failures is unacceptable. Count the number of software causes of hazardous conditions for each subsystem; subsystems where more than 25% of causes are software-related have high software safety risk.

6 Propose **responses** to identified risks. What are the decisions and actions to be taken for each risk?

   Examples: Extend testing on Component X for one week. Revise the design process to account for software interaction with hardware components.

In the remainder of this paper, we apply these six steps to the hazard analysis process on the Constellation program.

3. CONTEXT OF THE CONSTELLATION PROGRAM

The Constellation program is a complex system of systems (see Figure 1). Each system contains multiple elements with numerous, complex hardware and software subsystems. Our research focuses on three spaceflight hardware projects: A, B, C – one at the system level and two at the element level. . Project A is developed by NASA while Projects B and C are being developed by contractor organizations. Software is a critical element in controlling the function of these systems, and the amount of software varies significantly in each project. The names of the projects are kept anonymous for confidentiality purposes. At the time of this case study, the projects were in the preliminary design phases: an extensive requirements phase had been completed for all projects, and an initial design was under development.

Analyzing and designing to mitigate software risk is supported by NASA Software, Reliability and Quality Assurance (SR&QA) personnel. SR&QA is a division within the Constellation program that provides guidance to safety engineers on the specific projects and participates in CSERP safety reviews. This division is comprised of NASA employees and contractors with expertise in hardware, software and mission assurance. Their challenge was to gauge the ongoing state of software safety during preliminary design to help guide safety and reliability design efforts.

The multitude of systems that comprise the Constellation program are developed by contracting organizations using a form of concurrent engineering [8] wherein multiple development activities (i.e. design, implementation, testing) occur in parallel. For SR&QA personnel, obtaining an accurate, program-wide picture of software safety risk is difficult across these multiple, independently-developing systems for a number of reasons:

- There are many development groups, each with their own reporting style for safety risks. Even though program-wide standards exist, each group has their own interpretation of how to address those standards.
- The NASA panel charged with overseeing system safety has limited resources and technical knowledge to fully understand all the implications of software safety. Although these experts have significant experience managing the development of rockets and spacecraft at NASA, applying NASA safety processes to software is relatively new.
- The safety engineers responsible for the systems sometimes have limited understanding of how to describe software safety risk to meet the requirements of NASA safety reviews.
- The rules for recording software risk in the safety tracking systems were only recently developed, resulting in no clear delineation between software-based risks and non-software-based risks.

At various times, checkpoint meetings are held by the Constellation Safety & Engineering Review Panel (CSERP), which acts as gatekeeper for development milestones. There are several milestones in the development process (e.g. system requirements review, preliminary design review, critical design review) with different requirements for the type of system and software safety analysis that must be performed. At each milestone, the development groups identify safety risks in system operation and design and create strategies (controls) for mitigating those risks. The CSERP reviews the risks and the operational or design strategies for mitigating these risks. The CSERP panel then approves the current design or requests changes to provide for better risk mitigation. As development progresses and the system matures, the analyses and designs become more specific and concrete. The primary responsibility of the CSERP is to ensure that all safety risks which could result in loss of life, loss of the vehicle, or loss of mission are identified and handled properly.

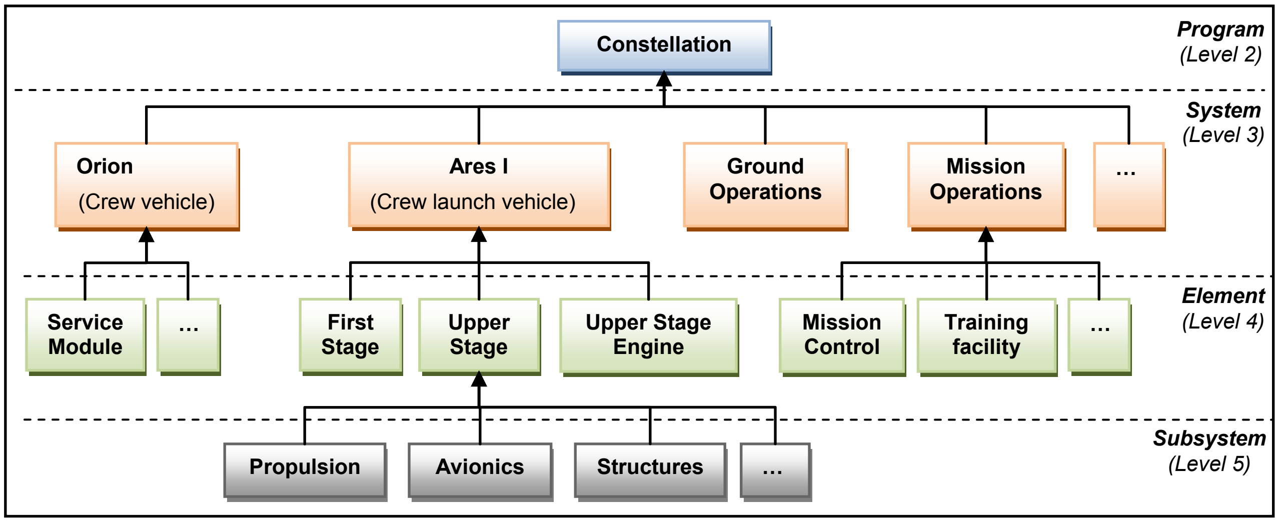

Figure 1. Constellation program hierarchy

4. APPLYING THE TPRM METHODOLOGY TO CONSTELLATION

In this section, we describe the application of the TPRM methodology to the Constellation program in order to support SR&QA personnel in managing software safety risk during the early design phases of the program.

4.1 Step 1: Insight Areas – The Hazard Analysis Process and Hazard Reports

Safety analysis in Constellation is vested in CSERP and its use of hazard analysis to drive actions. Thus, our study focused on the *hazard analysis* process used in the Constellation program. Hazard analysis is a top-down approach to system safety analysis [4]. The hazard analysis process for the Constellation program is mandatory for all projects and is defined by the Constellation Hazard Analysis Methodology process document (CxP 70038).

In hazard analysis, a *hazard* is any real or potential condition that can cause: injury, illness, or death to personnel; damage to or loss of a system, equipment, or property; or damage to the environment. An example of a hazard might be "Avionics hardware failure leads to loss of mission." The hazard is accompanied by a list of systems, elements and subsystems that cause or are affected by the hazard, a detailed description of the hazardous condition, and information regarding the likelihood of the hazardous condition occurring. All Critical severity hazards (severe injury, loss of mission, major damage) with a high or very high likelihood and all Catastrophic severity hazards (death/permanent injury, loss of vehicle) with moderate, high or very high likelihood were subject to CSERP review.

Hazards are described with several important properties:

- *Causes* – The root or symptomatic reason for the occurrence of a hazardous condition;
- *Controls* – An attribute of the design or operational constraint of the hardware/software that prevents a hazard or reduces the residual risk to an acceptable level;
- *Verifications* – A method for assuring that the hazard control has been implemented and is adequate through test, analysis, inspection, simulation or demonstration.

Figure 2 illustrates the conceptual organization of a hazard. Each hazard (e.g., engine failure) has one or more causes (e.g., failure with fuel line, software turns off the engine). Each cause has one or more controls that reduce the likelihood that a cause will occur or mitigates the impact should the cause be realized; controls often represent new requirements for the system (e.g., backup computers to account for software failures). Each control has one or more verifications (e.g. test, inspection, simulation or demonstration) to ensure that the control is appropriately implemented.

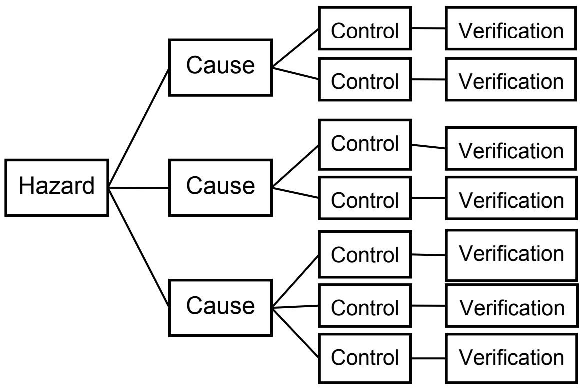

Figure 2. Hazard structure

It is important to note that, in this environment, software is never a hazard; hazards all represent physical events that may harm the mission. Component failure (e.g., degraded thruster performance) or outside events (e.g., hitting space debris, impact of weather, cosmic ray impact) may impact a mission, but software itself is not a hazard. However, software, as well as human error or component failure, can certainly *cause* a hazard (e.g., the software shutting a fuel valve at the incorrect time).

In the Constellation program, all hazards and their associated causes, controls and verifications are stored in a database called the *Hazard Tracking System* (HTS). Each such hazard is stored as a *Hazard Report* (HR) in the HTS. These process artifacts are rich in safety information and provide insight into areas of technical risk in the Constellation systems. They are also evidence of how the hazard analysis process is applied on the different projects.

An important note: our evaluation focuses on software *safety* risk. Safety, in the Constellation program, does not include software *security*. Software security on the Constellation program is handled a by separate organization charged with hardening the software systems against malicious attack and assisting in secure software design.

4.2 Step 2: Measurement Opportunities in Hazard Reports

We next identify measurement opportunities to help quantify software safety risk based on the rich safety information captured in the hazard reports. One measurement opportunity for quantifying technical risk is to measure the number of hazards, causes, controls and verifications that involve software. This helps to provide an overall picture of the prevalence of software involvement in hazardous conditions. These measurements can be sorted according to other hazard report data, such as the affected subsystems and missions phases in which the hazard is relevant.

To quantify process risk, we can examine the content and specificity of the software causes, controls and verifications to determine if they adhere to the guidelines set forth in the Constellation Hazard Analysis Process document (CxP 70038). Specificity in hazard causes is important for developing concrete, verifiable controls. A lack of specificity in the definition of causes indicates a risk that the cause has not been adequately identified and evaluated for control strategies.

Another measurement opportunity involved so-called *"generic" software hazard reports*. Each of the projects had 1-2 "generic" software hazard reports, which describe only the procedures for how software should be developed, but do not describe specific design or behavior. Some software causes had a single control referring only to these "generic" reports rather than a specific design attribute. These controls represent risk in that there is no objective verification that a software cause has been controlled by adhering to the software process.

Another measurement opportunity involves counting the number of *transferred* causes, controls and verifications. Transfers are a reference to a cause, control or verification in another hazard report. Transfers imply that the cause, control or verification is fully described in the other hazard report and does not need to be repeated. For example, a structural collapse will impact nearly every system in a hazard. Rather than list causes and controls for structural collapse in every hazard report, it is handled in its own report that is referred to by the other hazards. During system implementation and test, all transfers must be verified for a hazard reports to be considered "closed." Verifying transfers is a manual, labor intensive process and is at risk when transfer references are not kept up to date. Thus, transfers themselves are a measurement opportunity as they can represent both technical and a process risk.

4.3 Step 3: Readiness Assessment Questions

We developed a number of readiness assessment questions while exploring the measurement opportunities and offer a sample below. It is important to note that answering these questions is an iterative process, as answering one question may lead to others.

Can we access the hazard tracking system and hazard reports?

The HTS was made available to us by NASA personnel. Access to the process artifacts was a necessary precondition to any measurement.

Is the content of the hazard tracking system up to date?

The hazard tracking system only contained up-to-date hazard reports for one project (Project A). The hazard reports from other projects were obtained from a database containing materials from CSERP review meetings. Project B's hazard reports were in the HTS but not visible as the development company did not want to release "intermediate" versions of the hazard reports. Project C's hazard reports were created prior to the development of the hazard tracking system itself. Since Project B and C's hazard report were not in the HTS, we could not leverage the querying capabilities of the HTS and had to spend additional effort collecting all of the hazard reports for the two systems.

Are the cause, control and verification data complete enough for analysis?

For the NASA engineers writing the hazard reports, the goal of hazard analysis in the preliminary design phase was to identify and describe all causes and to develop preliminary controls. Verifications were not yet specified nor required. As such, we could not measure software verifications. Most causes and controls were specified and thus could be analyzed, though some were still works in progress and had a "To Be Determined" placeholder.

4.4 Step 4: Define Goals and Questions

To describe our goals for evaluating software safety risk, we use the Goals Questions Metrics (GQM) model [2]:

1. Analyze a sample of the hazards reported for Projects A, B and C in order to characterize them with respect to the prevalence of software in *hazards*, *causes*, and *controls* from the point of view of NASA quality assurance personnel in the context of the Constellation program.

2. Analyze the *software causes* in a sample set of hazard reports for Projects A, B and C in order to evaluate them with respect to the specificity of those software causes and hazards from the point of view of NASA quality assurance personnel in the context of the Constellation program.

Goal 1: Quantify the prevalence of software in hazards, causes and controls

Goal 1 is useful to SR&QA personnel in the early design phases to understanding the risk analysis effort required to adequately control software safety risk and also to identify systems and subsystems that involved more software risk than others.

We first define what it means for software to be involved in a hazard, cause or control. We define a *software hazard* as a hazard that contains one or more software causes. A *software-related hazard* is a hazard where software is either one of the causes or software is in one or more of the controls. We are interested in software-related hazards because, even though software may not be a direct cause of a hazard, software that is part of the control can be faulty and cause a subsequent hazard (as in the Therac-25 disasters [6]). Software hazards are a proper subset of software-related hazards. Both software hazards and software-related hazards may include hardware causes and controls as well. A *software cause or control* contains one or more of the following in its description: FCSW (flight computer software), FC (flight computer), specific CSCIs, or used the word "software." CSCIs (Computer Software Configuration Items) are software programs (e.g. Guidance Navigation & Control, Vehicle Management) that are involved in the commanding of sub-systems (e.g. avionics, propulsion) and would represent Level 6 in Figure 1.

These definitions help us formulate the following questions regarding the role of software in system safety:

1. What percentage of hazard causes are software causes?
2. What percentage of the hazards is software–related?
3. What percentage of hazard causes are non-software causes (e.g., hardware, operational error, procedural error) with software controls? These causes represent potentially "hidden" software risks. For example, if a software control is monitoring a hardware condition, then if the monitoring software fails there is a risk that the monitor will fail to detect an actual subsequent problem or the software may send erroneous status messages. Thus, the software can again be the cause of a hazardous condition.
4. What percentage of all non-software causes contains software controls?
5. What percentage of all causes contains software controls?
6. What percentage of causes is transferred?
7. What percentage of controls is transferred?
8. What percentage of the non-transferred hazard controls is specific software controls, i.e. describe software behavior or design?
9. What percentage of non-transferred controls are references to "generic" software controls?

Goal 2: Evaluate the specificity of software causes

Goal 2 assists SR&QA personnel in the early design phases by identifying software hazards and software causes that require additional work on the part of the safety engineers. Furthermore, hazard reports mature over time, and the evaluation of Goal 2 enables SR&QA personnel to track the maturation of software causes as the systems approach their quality milestones.

It is important to remember that this analysis was conducted as the projects entered their *preliminary design reviews*. The Constellation hazard analysis process only required that hazard *causes* meet certain detail and specificity criteria. Goal 2 is in some respects a proxy for SR&QA and CSERP personnel, who must understand the cause of a hazardous condition as described in the hazard reports. Software causes are evaluated according to their *specificity*, which is prescribed by the hazard analysis process. We do not evaluate the quality of the causes, as this requires significant domain knowledge and is one of the core purposes of CSERP review.

1. What percentage of software causes is *well-specified* according to the Constellation hazard analysis methodology requirements?
2. What percentage of software causes is *partially-specified?* These causes lack certain pieces of information needed to evaluate their quality.
3. What percentage of software causes is *generically-defined?* A "generic" cause (e.g. "the software fails") is not specific enough to identify any control strategy.

We define metrics for evaluating the specificity of causes based on the hazard analysis process documentation in Section 4.5.2.

4.5 Step 5: Develop Measures, Models and Interpretations

In this subsection, we describe the measures and models created for each of the goals in the previous section and describe our procedures for collecting the data.

Goal 1: Quantify the prevalence of software in hazards, causes and controls

By obtaining basic counts of software hazards, software causes and software controls, we were able to answer the questions posed in the previous section.

Measures and models

The measures were based on the counting of software-related hazards, software causes, software controls and transferred causes and controls. The following basic metrics were collected:

- The total number of hazards, causes and controls
- The number and percentage of software-related hazards
- The number and percentage of software causes
- The number and percentage of software controls
- The number and percentage of transferred causes
- The number and percentage of transferred controls

While no specific thresholds were set by NASA personnel regarding these counts (i.e. "what percentage amount of software controls is too high?"), the information can be used to identify subsystems with the most software risk that may require more software verification effort. Similarly, there are no pre-defined thresholds for acceptable or unacceptable amounts of transferred causes and controls. Part of this research effort is to establish a baseline for these measures to inform future projects.

Analysis procedure

A total of 154 hazard reports were analyzed for the three Constellation systems: 77 in the Project A, 57 in the Project B, and 20 in Project C. We first identified software and non-software causes and controls in the hazards reports. The analysis was performed manually by the first and third authors. The researchers performed the analysis on approximately 30% of the hazards then compared notes to refine the procedure. The remaining 70% were evenly distributed for analysis. The first author then verified the analysis data in a second iteration through all of the hazard reports.

The analysis of each hazard report was performed manually by reading the text of the causes and controls with the following steps:

1) Each cause from a hazard report was entered in a separate row of an Excel spreadsheet (see Table 1).
2) Each control in the hazard report was listed in a column in the spreadsheet. The controls for each cause were marked as a

software control (green), a non-software control (blue), a control that transferred to another hazard report (orange), or a transfer to a "generic" software hazard report (yellow).

3) The causes were marked as either a software cause (green) or a non-software cause (white). Causes for which *all* controls were transferred were marked red and excluded from further analysis under the assumption that the cause was controlled by the transferred hazard report(s).

The classifications of causes and controls were then counted for each hazard report and recorded in a separate worksheet (see Table 2 for an example). These data were used to compute summary statistics across all hazard reports and to answer the questions posed in Section 4.5. The "causes" column is the total number of causes listed in the hazard report, and the "active causes" column is the number of non-red causes in the cause-control matrix.

Table 1. Cause-control matrix example

Hazard Report	**Cause**	**Controls**																
		1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***
HR1	1	🟦																
	🟩 2		🟩													🟩	🟨	
	3			🟦	🟦	🟦	🟩											
	🟥 4							🟫										
	🟥 5								🟫									
	6									🟦	🟫							
	7											🟦	🟦					
	🟥 8													🟫				🟨
	🟥 9														🟫			

□	🟩 SW	🟦 Non-SW	🟫 transferred	🟨 Transferred to generic SW HR

Table 2. Example tabulation of causes and controls

Hazard Report	**Causes**	**Active causes**	**Transferred causes**	**SW causes**	**Non-SW causes with SW controls**	**Controls**	**SW controls**	**Transferred controls**	**Generic SW transfer**
HR1	9	5	4	4	1	12	4	3	1
HR2	14	10	4	0	6	33	7	12	3

Table 3. Summary metrics: Measuring the prevalence of software in hazards, causes and controls

	Question	**Project A**	**Project B**	**Project C**
1	What percentage of the hazard causes are software causes?	15%	12%	17%
2	What percentage of the hazards is software-related?	49%	67%	70%
3	What percentage of hazard causes are hardware causes with software controls?	14%	11%	-
4	What percentage of hardware causes has software controls?	16%	13%	-
5	What percentage of the causes has software controls?	29%	23%	-
6	What percentage of causes is transferred?	31%	23%	36%
7	What percentage of controls is transferred?	22%	11%	-
8	What percentage of the non-transferred hazard controls are specific software controls?	12%	14%	
9	What percentage of the non-transferred hazard controls are references to "generic" software controls?	5%	2%	-

Interpretation

From these data, we calculate the metrics necessary to answer the questions from Section 4.5 that help quantify the importance of software with respect to system safety. Table 3 presents the metrics data corresponding to the questions from Section 4.5 (the raw metrics used to compute these data are provided in the Appendix). These data demonstrate that although a small percentage (12-17%) of hazard causes are software causes, the percentage of hazards that are either caused by software or are controlled by software in much higher (49-70%). This indicates that software is a safety-critical aspect of the overall system and over half of all hazard reports are software-related.

Note that while 49% of Project A's hazard reports are software-related, 67% of Project B hazard reports and 70% of Project C hazards are software related. This disparity can be a consequence of the characteristics of the three systems, an indication of how the three projects organize the subjects of the hazard reports differently, or a combination of these. In all three systems, the importance of software clearly demonstrates the need for a strong software development process with adequate control and verification. Additional discussion and interpretation of these metrics can be found in [5].

It is clear that software plays a significant role in the safety of the Constellation program. However, there is variable precision in the counting of software hazards, causes and controls; the guidelines for reporting hazards are open to interpretation and each group reported and scoped hazards differently. The lack of a uniform structure for reporting software-related hazards represents a risk in that the non-uniform structure inhibits a consistent, general methodology for software risk assessment based on the hazard reports. Furthermore, the number of software and software-related hazards is likely greater than shown as there may have been software causes and controls that were not identified as such.

From the point of view of SR&QA personnel, it is difficult to track each hazard cause and control to its source, and overall traceability becomes more difficult. These traceability problems are a form of process risk and are expanded upon in Section 4.6.

Goal 2: Evaluate the specificity of software causes

We derived an initial set of software cause metrics that can be applied to measure the specificity of software causes. The metrics are based on the requirements for describing software causes set forth in the Constellation Hazard Analysis Methodology process document (CxP 70038). Additional input was drawn from the NASA Software Assurance Standard (NASA-STD-8739.8), the NASA Software Safety Standard (NASA-STD-8719.13B), and the NASA Software Safety Guidebook (NASA-GB-8719.13). These metrics were developed and approved with feedback from NASA SR&QA responsible for software assurance on the Constellation program.

Metrics and models

For the preliminary design review milestone, the Constellation hazard analysis methodology requires that software is defined to the level of Computer Software Configuration Items (CSCIs). CSCIs can be used in the analysis of relationships between components and specifying the safety-critical events, commands and data. Describing software causes at the CSCI level enables the hazard analyst to identify specific design elements that satisfy the requirement for controls.

We measure the minimal specificity of software causes based upon the existence of three attributes in the cause description: (1) which CSCI may fail its intended operation (*origin*), (2) the erroneous behavior for this software component (*erratum*), and (3) the impact of this erroneous behavior on the system (*impact*). The metrics are defined as:

1. For each hazard report, what are the number and percentages of L1, L2 and L3 causes where L1, L2 and L3 are defined as:
    - **L1**: a specific software cause or sub-cause for a hazard, where a specific software cause *must* include all of the following:
        - Origin – the CSCI that fails to perform its operation correctly
        - Erratum – a description of the erroneous command, command sequence or failed operation of the CSCI
        - Impact – the effect of the erratum where failure to control results in the hazardous condition, and if known, the specific CSCI(s) or hardware subsystem(s) affected
    - **L2**: a partially-specified software cause or sub-cause for a hazard, where a partially-specified software cause specifies one or two of the origin, erratum or receiver at the CSCI/hardware subsystem level.
    - **L3**: a generically defined software cause or sub-cause for a hazard, where a generically-defined software cause does not specify the origin, erratum or receiver at the CSCI/hardware subsystem level.
2. For each system, what are the number and percentages of La, Lb, Lc, Ld and Le hazards where La-Le are defined as:
    - **La**: All software causes and sub-causes in a hazard are L1
    - **Lb:** all software causes and sub-causes in a hazard are L1 except for a single L3
    - **Lc**: Software causes and sub-causes are a mix of L1, L2 and L3 with at least one L1
    - **Ld**: All software causes and sub-causes are either L2 or L3 with at least one L2
    - **Le**: All software causes are L3

A low hazard rating (e.g., Ld and Le) may indicate that there is a risk of not being able to mitigate the software risk associated with these hazards. A high rating (e.g., La and Lb) more likely indicates that the development team fully understands the risk and has addressed it appropriately. The overall hazard ratings provide a top level view of the maturity of software cause specificity in a subsystem or mission element.

We note that *these ratings do not measure the quality or the completeness of the software cause and control analysis*; these ratings only reflect the specificity of the information captured in the hazard reports. We believe that these ratings likely indicate risk where insufficient specificity has been provided to identify the software-based cause of a hazardous condition within the hazard report. Insufficient specificity probably indicates that the problem is not well understood, unless further details are included in supporting documentation. However, unless such supporting information (and the necessary context and expertise to interpret it) are maintained with the hazard report, there is risk that information will be lost.

Analysis procedure

The above metrics were applied to 385 software causes and sub-causes. Software causes were identified in the analysis for Goal 1. Many software causes contained a number of "sub-causes." Sub-causes were identifiable by either: 1) explicit enumeration in the cause description by the hazard report author; or 2) separate paragraphs describing errors by different CSCIs. Because sub-causes described different software behaviors, each was measured for its specificity. The first author and third author applied the cause ratings to the 385 software causes and sub-causes in an initial iteration, and the first author checked for consistency in a second iteration of all the causes. The first author then computed the software cause and hazard metrics.

Table 4. Project A software specificity metrics

Cause ratings		
L1	65	50%
L2	26	20%
L3	38	29%

Hazard ratings		
La	5	18%
Lb	7	25%
Lc	7	25%
Ld	3	11%
Le	6	21%

Table 5. Project B software specificity metrics

Cause ratings		
L1	64	38%
L2	68	40%
L3	37	22%

Hazard ratings		
La	3	8%
Lb	1	3%
Lc	14	38%
Ld	13	35%
Le	6	16%

Table 6. Project C software specificity metrics

Cause ratings		
L1	0	0%
L2	41	71%
L3	16	29%

Hazard ratings		
La	0	0%
Lb	0	0%
Lc	0	0%
Ld	12	86%
Le	2	14%

Interpretation

For all 3 projects, the causes were rated at least Level L2 for 70-78% of all causes. There are also noticeable differences between projects. Project A had a greater proportion of well-specified software causes than Projects B and C at the time of analysis. Project B had a large portion of software-causes that could be considered "in work," and thus one would expect the distribution to shift to the higher end of the scale as work progresses. Project C, however, was non-specific in terms of software causes. In general, the software causes in Project C had very specific descriptions of the impact of a software failure on the non-software, but little was described in terms of what caused the software to malfunction or how the software error manifested beyond stating that "the software fails." Although this data is only from PDR, project C has significantly less specificity that the other two.

Part of the CSERP review process is to further refine the specificity of hazard reports over time, particularly in subsequent development phases. These current figures provide an important baseline for CSERP and SR&QA personnel to monitor the progress of hazard report specificity over time.

Once again, we note that *these ratings do not measure the quality or the completeness of the software cause and control analysis*; these ratings only reflect the specificity of the information captured in the hazard reports. We believe that these ratings reflect risk where insufficient specificity has been provided to identify the software-based cause of a hazardous condition within the hazard report. Insufficient specificity may indicate that the problem is not well understood.

4.6 Step 6: Responses to Identified Risks

In the process of developing this data, we uncovered a number of potential process risks for the program with regard to how software-related hazards are reported. Some of these risks were interpreted from the metrics above, while others were discovered during data analysis and vetted by SR&QA.

Risk 1 *– Lack of consistency in structuring hazard report content, causes and control descriptions impairs understanding.*

All hazard reports in the Constellation program follow a standard template, but the content of the hazard reports, cause descriptions, and control descriptions differed substantially between the three programs and between hazard report authors within the same program. In some cases, the unstructured text creates risk that the CSERP may not be able to fully understand the risks detailed in the hazard even with supporting materials.

During preliminary design, safety engineers are still developing first versions of hazard reports and becoming familiar with the expectations of CSERP and the requirements of the software safety process. This risk has abated over time as CSERP and SR&QA personnel have worked closely with safety engineers to form a uniform expectation for hazard report content. These experiences are also being used to recommend improvements to NASA process documentation and training materials.

Risk 2 *– Lack of consistent scope in causes and controls impairs risk assessment.*

Related to Risk 1, there is a lack of uniformity in scoping software causes and controls between programs or between hazard reports within programs in some cases. A cause reading "Generic avionics failure or software flaw causes improper operation of control thruster" certainly involves software, but it is not scoped to a particular software component as required by NASA procedure.

Much of this risk can be attributed to unfamiliarity with describing software risk in hazards and misunderstanding the expectations of the CSERP board. This risk has also abated over time, yet remains present in some hazard reports. SR&QA personnel are conducting workshops with project safety engineers to educate them further on describing software risk. We have also provided a two-page "user guide" with examples of how safety engineers can specify software causes of hazards that has been well-received by SR&QA personnel. Furthermore, NASA technicians are considering changes to the hazard tracking system to enable safety engineers to mark software causes, controls and verifications as involving software.

Risk 3 *– "Lumped" software causes and controls impede verification.*

Many hazard reports placed all software causes and most software controls under a single cause labeled "Software-based error." In many cases, this cause had a single control with multiple pages of software design and operational information. This large control then had a single verification. This single control, while highly detailed, presents risk in that software design and behaviors will not be individually verified.

As with the previous risk, CSERP and SR&QA personnel are working closely with project safety engineers to "modularize" the description of software causes controls instead of treating software as a single black-box entity. A constant challenge faced by CSERP, SR&QA and safety engineers is determining when differentiating complex hardware and software functionalities into multiple causes and controls is appropriate. Complex causes and controls introduce risk that some individual risks may not be well understood. However, creating controls also entails significant additional verification effort that may yield little return if the cause/control was largely covered elsewhere.

Risk 4 *– Incorrect references to hazard reports, causes and controls impair traceability.*

A number of references to missing or incorrect hazard reports, causes or controls were observed. The most substantial risk is that a cause may not be adequately controlled when one or more of its controls are transferred to an incorrect or missing hazard report, cause, or control. NASA is currently deploying improved functionality in the HTS to allow safety engineers to create explicit references to other hazards, causes, controls and verifications in the hazard reports. This functionality will be backed by automated verification and bookkeeping.

Risk 5 *– Sub-controls dissuade independent verification and add overhead.*

Many HRs have controls that contain enumerated "sub-controls." Greater confidence in the control may be gained by verifying the sub-controls independently. Furthermore, additional risk is introduced in that references to sub-controls may become lost or incorrect as these references must necessarily be manual instead of taking advantage of the technology available in the hazard tracking system. As in Risk 3, CSERP and SR&QA personnel are working closely with safety engineers to determine the best methods for separating out and managing the overhead associated with complex controls.

Risk 6 *– Ubiquity of transferred causes and controls may mask software risk.*

Across the projects, 23-31% of causes and 11-22% of controls were transferred. While necessary and appropriate in documenting hazards, transferred causes and controls represent added risk. The applicability of transferred causes and the adequacy of transferred controls must be re-evaluated in their original context whenever any changes are made to the causes or controls. Furthermore, additional bookkeeping is necessary to ensure that the references to hazard reports, causes and controls are up to date (see Risk 4). Transferred causes and controls also make it difficult to understand the impact of software.

Stronger tool support (as described in Risk 4) enables better traceability and bookkeeping, but also enables analysis that can be used to quantify software risk. Coupled with marking causes and controls as software (as described in Risk 2), the HTS tool could then report the number of software causes and controls by automatically resolving dependencies between hazards.

5. SUMMARY AND FUTURE WORK

By applying the TPRM methodology, we assisted NASA SR&QA personnel in identifying areas of software safety risk in the early design phases on the Constellation program using hazard analysis artifacts. Furthermore, as Section 4.6 demonstrates, we were able to identify six specific process risks in the application of the hazard analysis process that may result in developing unreliable software. We are working with NASA SR&QA personnel in an ongoing effort to educate NASA safety engineers on describing software safety risk, to improve NASA process documents and training materials, and to provide tool support to the software safety process.

These results were similar to a previous case study of the TPRM methodology applied to the safety analysis process on a large Department of Defense system development [3]. In that case study we identified a similar series of risks:

- *Software Hazard Identification.* Safety-related requirements were not identified as such and many hazard *controls* were not properly identified as software-related safety requirements.
- *Hazard Traceability*. The HTS does not provide sufficient linkages among the requirements documentation system, the test plan, or to the defect tracking system.
- *Data Integrity*. Hazards, causes, and controls may not be described in sufficient detail to be understood and verified.
- L*evel of* R*igor (LoR)*. There was difficulty in differentiating among different levels of rigor for the various software safety requirements and identifying, assigning, and tracking the appropriate LoR to specific software components that implement the safety-related requirement. Lack of proper LoR differentiation can lead to inadequate attention on high-risk hazards or too much attention on low-risk hazards.

These risks are very similar to the risks we found in the NASA case study, and may be indicative of process risks experienced by many large software engineering projects being developed by many organizations. This indicates that simply defining a development process is not sufficient to identify safety (or any other kind of) risk. Management, measurement, and feedback of the process being used is as important as defining a proper process in the first place.

The major risk in systems development is the assumption that good development practices will result in good software. The value of the TRPM methodology is contingent on the studied process being the right process to achieve the desired product characteristic. We are assuming that adherence to the defined process, be it waterfall, agile, or whatever is appropriate in a given situation, will result in lower risk of safety problems. We do not say that adherence will guarantee a quality product, but do assume that non-adherence will increase the risk of failure.

This means that we have to apply the TPRM methodology to a larger number of case studies, environments and organizations to both show that the methodology does find safety risks, and to understand how prevalent these risks seem to be across the industry. To date the process works, but two anecdotes do not a theory make. We plan on continuing this in other environments.

6. ACKNOWLEDGMENTS

This research was supported by NASA OSMA SARP grant NNX08AZ60G to the Fraunhofer CESE. We would like to acknowledge the help of Karen Fisher and Risha George at NASA Goddard Space Flight Center for providing us support and access to people and artifacts of the Constellation Program.

7. REFERENCES

[1] D. Ahern, A. Clouse, and R. Turner,Cmmi® distilled: a practical introduction to integrated process improvement,

third edition, Third edition, Addison-Wesley Professional, 2008.

[2] V. Basili and D. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, 10(3):728-738, Nov. 1984.

[3] V. Basili, F. Marotta, K. Dangle, L. Esker, and I. Rus, "Measures and Risk Indicators for Early Insights Into Software Safety," *Crosstalk*, 21(10):4-8, Oct. 2008,

[4] Federal Aviation Administration, "System Safety Handbook", accessed October 15, 2010, http://www.faa.gov/library/manuals/aviation/risk_management/ss_handbook/, updated May 21, 2008.

[5] L. Layman, V. R. Basili, M. V. Zelkowitz, "The Role and Quality of Software Safety in the NASA Constellation Program", Fraunhofer CESE Technical Report #10-101, http://www.fc-md.umd.edu/Publications/TR/Safety-metrics_TR_10-101.pdf, June 2010.

[6] N. G. Leveson and C. S. Turner, "An investigation of the *Therac-25* accidents," *IEEE Computer*, 26(7):18-41, doi:10.1109/MC.1993.274940, July 1993.

[7] B. Nuseibeh, "Ariane 5: Who Dunnit?", *IEEE Software*¸14(3):15-16, May/June 1997.

[8] B. Prasad, Concurrent Engineering Fundamentals – Integrated Product and Process Organization, Prentice Hall, Upper Saddle River NJ, 1996.

APPENDIX

Table 7. Project A - hazard table

	Non-software cause	At least 1 software cause
no software control	39 *(51%)*	0 *(0%)*
at least 1 software control	10 *(13%)*	28 *(36%)*
Total	77	

Table 8. Project A - cause table

	Non-software cause	Software cause
no software control	393 *(71%)*	0 *(0%)*
at least 1 software control	76 *(14%)*	85 *(15%)*
Transferred causes	252	
Total	806	

Table 9. Project A - control table

	N	% of total	% of non-transferred
Non-software	1603	64%	82%
Software	243	10%	12%
Generic software controls	105	4%	5%
Transferred controls	566	22%	-
Total	2517	100%	

Table 10. Project B - hazard table

	Non-software cause	At least 1 software cause
no software control	19 *(33%)*	0 *(0%)*
at least 1 software control	1 *(2%)*	37 *(65%)*
Total	57	

Table 11. Project B - cause table

	Non-software cause	Software cause
no software control	398 *(77%)*	0 *(0%)*
at least 1 software control	57 *(11%)*	62 *(12%)*
Transferred causes	155	
Total causes	672	

Table 12. Project B - control table

	N	% of total	% of non-transferred
Non-software	1799	75%	84%
Software	298	12%	14%
Generic software controls	37	2%	2%
Transferred controls	265	11%	-
Total	2399	100%	

Table 13. Project C - hazard table

	Non-software cause	At least 1 software cause
no software control	6 *(30%)*	0 *(0%)*
at least 1 software control	0 *(0%)*	14 *(70%)*
Total	20	

Table 14. Project C - cause table

	Non-software cause	Software cause
no software control	275 *(81%)*	0 *(0%)*
at least 1 software control	9 *(3%)*	57 *(17%)*
Transferred causes	194	
Total	535	