The Role and Quality of Software Safety in the NASA Constellation Program

Lucas Layman, Victor R. Basili, Marvin V. Zelkowitz

Fraunhofer Center for Experimental Software Engineering
College Park, Maryland 20740

Technical Report 10-101
June, 2010

Fraunhofer USA

Center for Experimental
Software Engineering, Maryland

For Further Information, Contact:
Fraunhofer Center - Maryland
5825 University Research Ct.
Suite 1300
College Park, MD 20740
http://fc-md.umd.edu

Abstract

For NASA quality assurance managers, obtaining an accurate, program-wide picture of *software safety risk* is difficult across the multiple, independently-developing systems in the NASA Constellation program. In this study, we create metrics that leverage one source of safety information, hazard analysis, to provide NASA quality assurance managers with information regarding the ongoing state of software safety. The goal of this research was two-fold: 1) to quantify the importance of software with respect to system safety; and 2) to quantify the level of risk presented by software in the hazard analysis.

We examined 154 hazard reports created during the preliminary design of three major flight hardware systems in the Constellation program. To quantify the importance of software, we collected metrics based on the number of software-related causes and controls of hazardous conditions. To quantify the level of risk presented by software, and we created and applied a metric scheme to measure the specificity of software causes descriptions.

We found that 49-70% of hazardous conditions in the three systems could be caused by software or software was involved in the prevention of the hazard. We also found that 12-17% of the 2013 hazard causes involved software, and that 23-29% of all causes had a software control. Furthermore, 10-12% of all controls were software-based. We applied our metrics for measuring the specificity of software causes and found that the results varied greatly between projects.

The application of our software specificity metrics identified risks in the hazard reporting process. Software causes are not consistently scoped, and the presence of software in a cause or control is not always clear. Furthermore, a number of traceability risks were present in the hazard reports that could impede verification of software and system safety.

1 Introduction

Development of large complex systems in aerospace, defense, energy and travel industries requires constant attention to safety risks. A *safety risk* is a risk whose cost is injury or the loss of life directly or through a chain of events. The analysis and mitigation of such safety risks significantly increases development time and cost. The task of controlling safety risk is further complicated when developing complex systems due to the amount of planning, assessment and communication required to manage multiple projects. The issue of controlling *software safety risk* has become a leading area of concern in systems development as many traditionally hardware-centric systems become more heavily reliant on software.

In this paper, we examine issues of measuring software safety in one such system – the multi-year, multi-billion dollar Constellation program at NASA. Software plays a significant role in many Constellation systems. Analysis, planning, and mitigation of safety risks (including software) pervade every phase of development. The multitude of systems that comprise the Constellation are developed by multiple contracting organizations using a form of *concurrent engineering* wherein multiple development activities (i.e. design, implementation, testing) occur in parallel. For NASA quality assurance managers, obtaining an accurate, program-wide picture

of software safety risk is difficult across these multiple, independently-developing systems for the following reasons:

- There are many development groups, each with their own reporting style for safety risks. Even though program-wide standards exist, each group had 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 they have managed the mostly non-software development of rockets and spacecraft at NASA, applying the NASA safety process to software was somewhat new.
- The safety engineers for responsible for the systems sometimes had limited understanding of how to describe software safety risk to meet the expectations of the NASA panel.
- 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.

We have created and applied software safety metrics that leverage one source of safety information, hazard analysis, to provide NASA quality assurance managers with information on the ongoing state of software safety. This metrics demonstrate the increased role of software as a control mechanism for safety risks, suggesting that increased emphasis on software safety analysis is warranted. The software safety metrics also yielded new guidelines for safety engineers to use when describing software safety risks.

The remainder of this paper is organized as follows. The remainder of this paper is organized as follows. In Section 2 we describe the research context and provide an overview of the hazard analysis process on the Constellation program. In Section 3 we describe the measures we capture on software risk described in hazard reports. In Section 4 we present the data findings of this study and in Section 5 we discuss these results and a several risks uncovered in the software safety reporting process. Finally, in Section 6 we give our conclusions on how such measures can best be used in this environment. Appendix A contains the "user guide" for specifying software causes derived during the course of this research.

2 Research context

The Constellation program is a complex *system of systems* (see Figure 1 for the Constellation program hierarchy). Each *system* contains multiple *elements*, each with numerous, complex hardware and software *subsystems*. Our research focuses on one system (labeled Project B) and two elements (Projects A and C). The scale and complexity of the Constellation program presents many challenges to NASA personnel in assessing the state of system safety and, in particular, software safety.

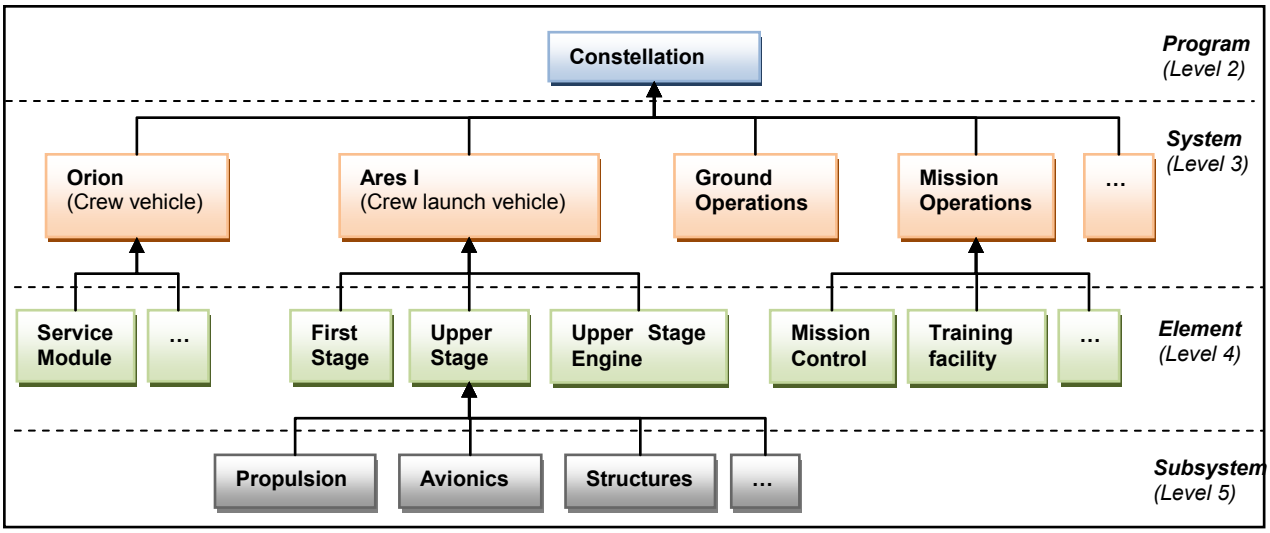

Figure 1. Constellation program hierarchy (abbreviated example)

2.1 Development process of Constellation

Development is overseen by NASA and performed by contract companies following a waterfall-like process. The development process used for the Constellation systems is a variation of *concurrent engineering*. Concurrent engineering [Prasad96] is an approach for dealing with the complexity of a large system development where multiple phases of development (i.e. analysis, design, implementation, etc.) occur in parallel with frequent checkpoints to make sure each phase is in alignment with the other phases. Coordination is a challenge in concurrent engineering as information needs to transfer among the various elements in order to distribute changes, new requirements, errors, and recommendations from one system/element to another.

Figure 2 presents an abstraction of the *safety monitoring* process model for Constellation elements. The development groups work on their own activities and use various databases for coordinating information among the groups. These include databases containing project requirements, safety hazards, defects, designs and other project management information. 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 (design or operational) for mitigating those risks. 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. Analyzing and designing to mitigate software risk is supported by NASA Safety, 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.

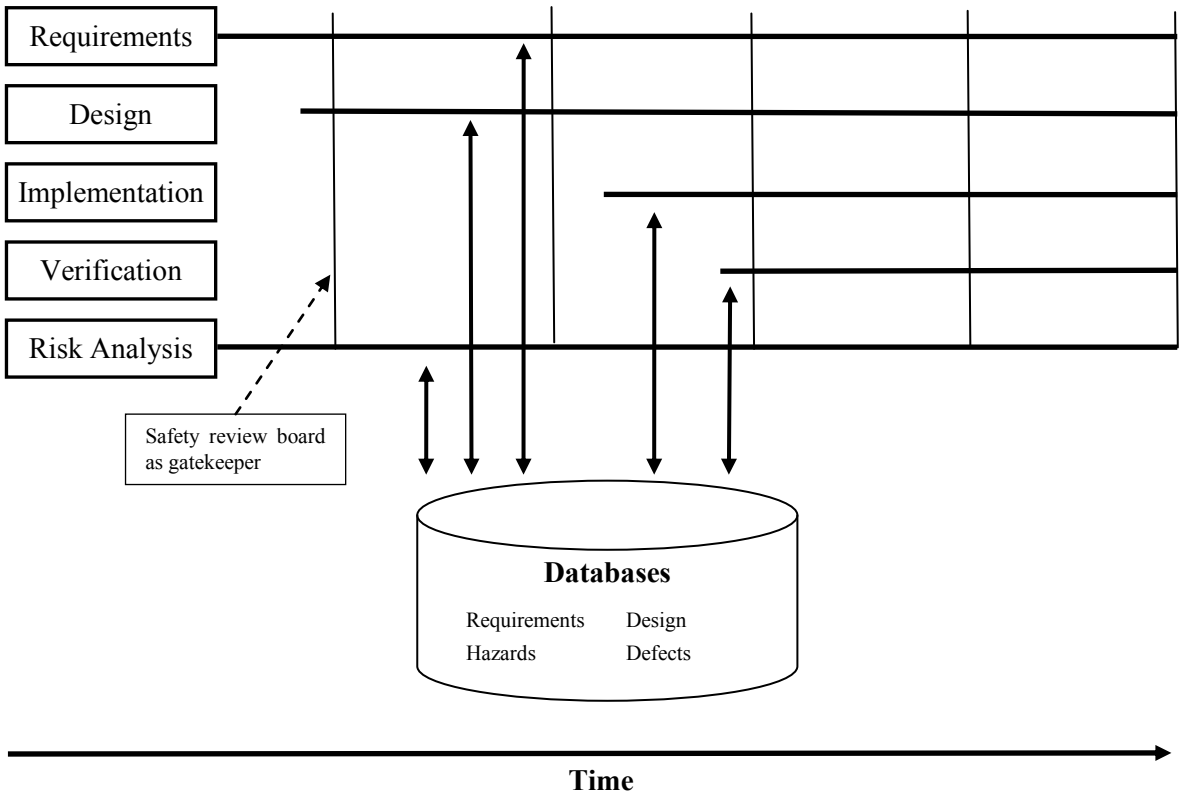

Figure 2. NASA safety monitoring process

2.2 Hazard Analysis in the Constellation Program

The purpose of the system safety process is to identify and mitigate hazards associated with the operation and maintenance of a system under development. System safety is often implemented through an approach that identifies hazards and defines actions that will mitigate the hazard and verify that the mitigations have been implemented. The *residual risk* is the risk remaining when a hazard cannot be completely mitigated. The goal of the system safety process is to reduce this residual risk to an acceptable level, as defined by the safety certifier. Cost is a consideration in determining the level of acceptable residual risk.

Our case study focused on the *hazard analysis* methodology in the Constellation program. Hazard analysis is a widely-used method in systems development that takes a top-down approach to system safety. In the Constellation program, hazard analysis is mandated by a Constellation process document (CxP 70038) and is complemented by other safety analysis methods, including failure modes and effects analysis (FMEA), fault-tree analysis (FTA), probabilistic risk assessment (PRA), quality audits, process and project metrics, and many more.

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 title is, "Avionics hardware failure leads to loss of mission." The hazard is accompanied by a list of systems, elements and subsystems affected by the hazard, a detailed description of the hazardous condition, and information regarding the likelihood of the hazardous condition occurring.

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 constraints of the hardware/software that prevent a hazard or reduce 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.

Each hazard (e.g., spacecraft engine fails) has one or more causes (e.g., failure of fuel line, software turns off engine, physical failure of engine). Each cause has one or more controls (e.g., backup computers to account for software failures), and each control has one or more verifications to ensure that the control is appropriately implemented (see Figure 3). 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.

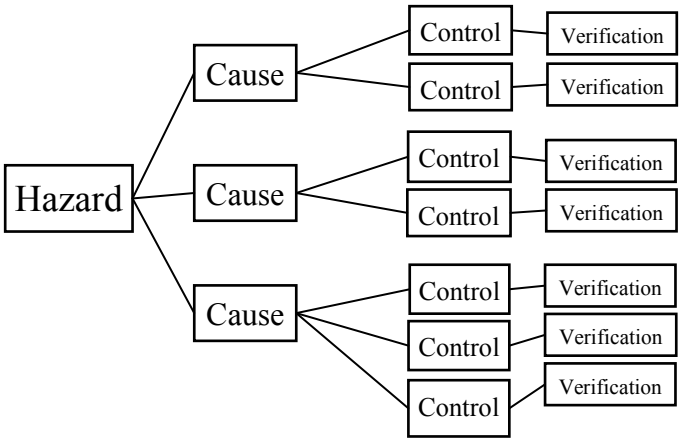

Figure 3. Hazard structure

Controls often represent requirements that need to be incorporated into the system. For example, if a buffer overflow causes a software component to overflow (e.g., the inertial guidance computer overflow in the Ariane 5 rocket failure in 1996 [Ariane5]), a possible control could be to monitor the values of the appropriate register and provide some mitigating action if it overflowed. This would represent a new requirement that the software developers would have to implement as part of building that software.

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). Therefore, the HTS contains only physical events, some of which are caused by software, and some of which are not.

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 [Leveson93]. Software hazards are a proper subset of the software-related hazards. Both software hazards and software-related hazards may include hardware causes and controls as well.

3 Research Method

We examine the Constellation hazard reports to assess the state of software safety risk and provide feedback into the software safety process. Our approach is based upon the premise that there is a relationship between the processes used during software development and the product's characteristics [Basili08]. That is, the implementation of a particular process will achieve particular goals with respect to the product. This leads to an assumption that a lack of process or lack of adherence to the process will lead to a risk of not achieving the anticipated product characteristics. While this assumption is sometimes true, it is often the case that failing to achieve the desired product characteristics is the result of a flawed process. By analyzing the execution of the process, we gain insight into whether appropriate processes are being performed and performed appropriately. Early visibility into the system safety process is important as safety risk mitigations typically affect system requirements and design; discovering and implementing safety risk mitigations late in the system development lifecycle can be cost-prohibitive.

The analysis of *process artifacts* available during development can help provide insight into potential risks of not achieving the desired product characteristics as well as insights into the failings of the process. As mentioned in Section 2.2, our case study focuses on *hazard analysis* in the Constellation program. Hazard reports are one artifact that is produced as part of the system safety process. We evaluate these hazard reports with respect to the software safety risks described within. We describe our goals for evaluating software safety risk using the Goals Questions Metrics (GQM) model [Basili84]:

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.

Software causes and controls are evaluated according to their *specificity*, whereas verifications of software controls are evaluated according to the level of *confidence or assurance in the verification*. Specificity in hazard causes is important for understanding 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 so as to be controlled. Each system we analyzed 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. Often, 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.

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.

3.1 Research goals

The goals of this study were derived and formalized using Basili, et al.'s Goal-Question-Metric approach [Basili84].

3.1.1 Goal 1: Quantify the importance of software with respect to system safety.

Goal 1 is useful to SR&QA personnel in understanding the risk analysis effort required to adequately control software risk and also to identify systems and subsystems that involved more software risk than others. We achieve this goal by classifying causes of a hazard and controls for these causes in hazard reports as either *software* or *non-software*. These data can then be used to answer a number of questions (see below). The following tables illustrate the categories of causes (Table 1) and hazards (Table 2).

Table 1. Cause categories

Cause Table	**Causes**		
Controls		**Non-software**	**Software**
	Non-software	(**A**) Non-software causes with no software controls	(**B**) Software causes with no software controls.
	Software	(**D**) Non-software causes with at least one software control	(**C**) Software causes with at least one software control

Table 2. Hazard categories

Hazard Table	**Causes**		
Controls		**Non-software**	**Software**
	Non-software	(**W**) Hazards with no software causes or controls	(**X**) Hazards with at least one software cause and no software controls
	Software	(**Z**) Hazards with no software causes and at least one software control	(**Y**) Hazards with at least one software cause and one software control

Questions:

1. What percentage of the hazards is software–related? That is, does software play a role in either causing or mitigating a hazard? Those hazards are represented by the entries in boxes X, Y and Z of Table 2. The percentage is given by (X+Y+Z)/(W+X+Y+Z).

2. What percentage of the hazard causes are software causes? This is represented by the right half of Table 1, or entries B and C. The percentage is given by (B+C)/(A+B+C+D).
3. What percentage of hazard causes are hardware causes with software controls, i.e., box Z in Table 2? This question indicates potentially "hidden" software risk. For example, if software, as a control, is monitoring a condition and is supposed to report a failure, if the monitoring software fails, even though the hardware is functioning correctly, 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. The percentage is given by D/(A+B+C+D).
4. What percentage of hardware causes contains software controls? D/(A+D)
5. What percentage of the causes contains software controls? (C+D)/(A+B+C+D)
6. What percentage of causes are transferred, i.e., another hazard report contains the control for this cause? Transferred causes and controls incur risk when traceability is not maintained.
7. What percentage of controls is transferred?
8. What percentage of the non-transferred hazard controls are specific software controls, i.e. describe software behavior or design? This may be a simple measure of the number of software related requirements that deal with safety critical software.
9. What percentage of non-transferred controls are references to "generic" software controls which document software process requirements, but do not describe specific software behavior or design for controlling specific causes? These controls represent risk in that there is no objective verification that the cause has been specifically controlled by adhering to the software process.

3.1.2 Goal 2: Quantify the level of risk presented by software.

Goal 2 assists SR&QA personnel 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. Goal 2 is in some respects a proxy for SR&QA and CSERP personnel, who must understand the origin of a hazardous condition (the cause) as described in the hazard reports. Software causes are evaluated according to their *specificity*.

As nearly all projects are in preliminary design, we focus on evaluating *causes* since only causes, which should be well-defined for a Phase I safety review prior to Preliminary Design Review (PDR). Based on the preliminary analysis of software-related hazards, we have derived an initial set of software safety metrics that can be applied to hazard reports. These metrics were developed using feedback from SR&QA personnel and using existing hazard reports as examples.

By PDR, the software is defined to the level of Computer Software Configuration Item (CSCI). We judge the minimal specificity of software causes based upon the existence of three attributes in the cause description: (1) which software component may fail its intended operation (*origin*), (2) what is the erroneous behavior for this software component (*erratum*), and (3) what impact does this erroneous behavior have on the system (*impact*)? We then define a metric to evaluate the specificity of a cause.

CSCIs (e.g. Guidance Navigation & Control, Vehicle Management) are more specific than sub-systems (e.g. avionics, propulsion) and would represent Level 6 and below in Figure 1, enabling the analysis of more specific causes and corresponding controls. Furthermore, 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.

Software-related causes and sub-causes may be described in a single cause in a hazard report or they may appear as multiple separate causes in a hazard report. From the software cause and sub-cause metrics for a given hazard, an overall hazard rating for each hazard report for software causes can be created.

Questions:

1. 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 (Computer Software Configuration Item) 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. 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 may (e.g., Ld and Le) indicate 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 what these ratings reflect may indicate 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. A lack of specificity may also mean that further details are located 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.

3.2 Data sources and context

A total of 154 hazard reports were collected for three Constellation systems: 77 in Project A, 57 in Project B, and 20 in Project C. These three systems were developed as separate projects, and each project performed their hazard analyses according to Constellation program guidelines. At the time of this research, the three systems were in the Preliminary Design Phase. During the Preliminary Design Phase, the safety engineers for each project met with the CSERP to evaluate, discuss, and mature the hazard reports. Additional description of the three elements is presented with the study findings below.

At the time of analysis, Projects B and C had completed their Phase I safety reviews, and the Project A Phase I review was ongoing. Projects A and B were pending additional "Phase I Software reviews" at the time of collection that lagged 2-3 months behind the development milestones of the rest of the system. The latest available versions of the hazards reports were used.

3.3 Analysis procedure

The bulk of the analysis was to identify software and non-software causes and controls in the hazards reports. An analysis of each hazard report was performed manually as follows:

1. The Hazard Report number from the PDF document was added to an Excel spreadsheet.
2. A row was added for each cause listed in the Cause Summaries section of the hazard report.
3. For each cause, each relevant control description was read and was marked as either a software control (green), a non-software control (blue), a control that transferred to another hazard report (orange), a transfer to another hazard report dealing exclusively with software (yellow), or controls with multiple sub-controls (grey, see b. below). The yellow controls were transfers to so-called "generic" software hazard reports, which describe only the procedures for how software should be developed, but do not describe specific design or behavior. A control was determined to be a software control if it described the behavior or design of FCSW (flight computer software), FC (flight computer), specific CSCIs, or used the word "software."
    a. Each control in the hazard report corresponds to a column in the spreadsheet (up to a maximum of 60 controls. The few with more than 60 controls were handled as special cases.
    b. Some software controls contained enumerated "sub-controls" that described separate software design and behavior. The parent control was marked as grey, and the sub-controls were listed in the spreadsheet in columns. These sub-controls were classified using the same scheme as in step 4.
4. The cause description for each cause was read and was 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 in the transferred hazard report(s). A cause was determined to be a software cause when software, FCSW (flight computer software), FC (flight computer) or specific CSCIs were mentioned in the cause description. Table 3 provides an example of the cause-control matrix for a hazard report.

Table 3. Cause-Control matrix for an example report

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

The classifications of causes and controls were then counted for each hazard report and recorded in a separate worksheet (see Table 4 for an example). These data were then used to compute summary statistics across all hazard reports and to answer the questions posed in Section 3.1.1. 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. Transferred controls (orange) and transferred generic software controls (yellows) separately.

Table 4. Example tabulation of causes and controls for HRs

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

4 Findings

These data were calculated for all of the hazard reports available for each project using hazard reports created just prior to or during Phase I safety reviews. The team reviewed a total of 77 hazard reports from Project A (Tables 5-7), 57 hazard reports from Project B (Table 8-10), and 20 hazard reports from Project C (Tables 11-13). This totals to 154 hazard reports, 2013 causes, and 4916 controls. Project C control data is not included as the project had very few specific controls defined within the hazard reports that could be counted.

Table 5. Project A hazard table

	Non-software cause			At least 1 software cause		
no software control	**W**	**39**	*51%*	**X**	**0**	*0*
at least 1 software control	**Z**	**10**	*13%*	**Y**	**28**	*36%*
Total	**77**					

Table 6. Project A cause table

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

Table 7. 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 8. Project B hazard table

	Non-software cause			At least 1 software cause		
no software control	W	19	*33%*	X	0	*0%*
at least 1 software control	Z	1	*2%*	Y	37	*65%*
Total	57					

Table 9. Project B cause table

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

Table 10. 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 11. Project C hazard table

	Non-software cause			At least 1 software cause		
no software control	W	6	*30%*	X	0	*0%*
at least 1 software control	Z	0	*0%*	Y	14	*70%*
Total	20					

Table 12. Project C cause table

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

4.1 Metrics summary

From these data, we calculate the metrics necessary to help answer the questions from Section 3.1.1 that help quantify the importance of software with respect to system safety (see Table 13).

Table 13. Findings for Goal 1: Quantify the importance of software with respect to system safety

	Question	Project A	Project B	Project C
1	What percentage of the hazards is software-related?	49%	67%	70%
2	What percentage of the hazard causes are software causes?	15%	12%	17%
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%	-

These data demonstrate that although a small percentage (12-17%) of hazard causes are software causes, the percentage of hazardous conditions that are either caused by software or are controlled by software in some way is high (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 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 subsystems, or it might simply be an indication of how the three elements organize the subjects of the hazard reports differently. The lack of a uniform structure for reporting software-related hazards inhibits a consistent, general methodology for software risk assessment based on the hazard reports. In all three systems, the importance of software clearly demonstrates the need for a strong software development process with adequate control and verification.

4.2 Software hazard and software cause ratings

To help answer Goal 2 in quantifying the level of risk presented by software, we applied the software cause and software hazard metrics described in Section 3.1.2 to the causes in the hazard reports. The L-metrics are applied to software causes and hazards with at least one software cause only. In many cases, software causes were broken down into "sub-causes". When this was done, each sub-cause received a rating, but the parent cause did not.

Table 14. Project A software cause ratings

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

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

Table 15. Project B software cause ratings

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

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

Table 16. Project C software cause ratings

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

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

Using the current software cause metric definitions, there are noticeable differences between the total system elements. Project A had a greater proportion of well-specified software causes than Project B and Project C at the time of analysis. Project B still 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 in the engine control unit beyond simply "the software fails."

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 may indicate 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. A lack of specificity may also mean that further details are located in supporting document. 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.

5 Results

In Section 3 we presented our two GQM goals for this study. We describe our conclusions for each goal in Section 5.1. We describe and discuss a number of risks observed in the hazard reports and their implications for software safety in Section 5.2.

5.1 Conclusions of GQM evaluation

GQM Goal 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.
It is clear from Section 4.1 and Table 13 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. 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 the quality assurance personnel, it is difficult to track each hazard cause and control to its source, and overall traceability becomes more difficult. In Section 5.2 below, we present our lessons learned about the nature of software in hazard reports, which directly addresses the issue of traceability and proposes some modifications to the development process to take this into account.

GQM Goal 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.*

Section 4.2 and Tables 14-16 provide the summary data for this goal. For all 3 projects, the causes were rated at least Level L2 for 70-78% of all causes. However, Project C had 0% with the highest rating of L1, whereas the other two projects had L1 ratings of 38% and 50%. 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.

Because this data was collected during the time for PDR, the process used by the Constellation Program did not require that controls and verifications be fully developed. For this reason, we were unable to fully characterize controls and their verifications for this study and it will have to wait for a later time.

5.2 Potential risks uncovered by the case study

In the process of developing this data, we have also recorded a number of potential risks for the program with regard to how software-related hazards are reported. A lack of completeness and uniformity in the information describing software risk (i.e. software causes and controls) creates a risk in itself. That is, software causes may not be thoroughly described and understood, and controls that involve software may be difficult to verify.

Maintaining traceability between causes, controls and verifications is essential for ensuring that all causes of hazardous conditions have a control in place and that this control has been verified. Our analysis was conducted on hazard reports that had passed or were in the Phase I review, thus,

complete description of controls and verifications were not required. However, traceability needs to be maintained between causes, controls and verifications during Phase I. We observed a number of risks associated with incomplete traceability in the hazard reports.

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

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. Considerable effort may then be required by the safety engineers to restructure the hazard report. Furthermore, the unstructured nature of some cause and control descriptions increases the risk that, when change to the hazard report occurs, information may become lost and plans for risk mitigation omitted. This risk is particularly true for software causes and controls, which tended to be unstructured descriptions of software behavior in many cases. In our evaluation, specific code words (i.e., "software," "FCSW," "computers") were searched in order to classify each hazard cause and control.

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 no uniformity in scoping software causes and controls between programs or between hazard reports within programs in some cases. A cause reading "Generic avionics/EPS failure or software flaw causes improper operation of control thruster" certainly involves software, but it is not scoped to a particular CSCI. Furthermore, each program appears to handle the impact of software differently. Project B reports software errors in every applicable hazard report, whereas Project A isolates the software functionality according to subsystem. Thus, while the impacts of software in Project B can be more directly observed by reading hazard reports, the effects of software in Project A are more transitive and may not be readily apparent. The difference in reporting style makes a uniform software risk assessment procedure across the program difficult to implement, thus making cross-element comparisons and summary status reports difficult to interpret.

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 (see Appendix A). 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.

For example, Project B hazard reports placed all software causes and most software controls under a single cause labeled "Software-based control error." The controls for these causes described the design and behavior of the CSCIs in high detail. A single control described the behavior of 4 or 5 CSCIs, covered several pages in the hazard report, and the individual behaviors were manually indexed within the control. This large software control had a single verification involving peer review, inspection, testing, and more. The control is large partly due to all software causes of the hazardous condition being described as a single cause. This single control, while highly detailed, presents risk in that software design and behaviors will not be individually verified. By contrast, many software causes and controls in Project A were targeted, focusing on individual behaviors of software with respect to sub-system function.

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. For example, Project B's "Software-based control error" causes typically had two controls: the first describing design intended functionality of the software, and the second which stated "In addition to the above-mentioned controls for software hazard causes, software also serves as a control for the following hardware hazard causes: […]." The latter control then listed specific causes from the hazard report and software controls for them. In many cases, the list of the causes was not accurate. For example, the text in one HR indicates that a particular control, say

control 20 is also a control for cause 11 – "human error," but inspection reveals that human error is actually cause 15. Such errors in the textual references to causes occur frequently in the controls for software-based control errors.

NASA technicians are 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." In Project B, these sub-controls were observed in every control for "software-based control error" causes. Instances were also found in a few HRs for Project A, where all controls (both non-software and software) for a cause were listed under one control. As discussed above, manually indexed sub-controls impede independent verification and/or require additional manual bookkeeping to ensure that the correct references are made between controls and their verifications. We observed that most controls that contain sub-controls only have one verification strategy even though multiple elements of design, functionality, or behavior may be described in the sub-control. 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). Finally, transferred causes and controls make it difficult to understand the impact of software. If software, by transitivity, is a cause of a hazardous condition, then the likelihood of the hazardous condition occurring may be higher than documented in the hazard report.

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 comprehensively the number of software causes and controls by automatically resolving dependencies between hazards.

6 Additional case study notes

We analyzed 154 hazard reports from the preliminary design phases of three flight hardware projects in the NASA Constellation program. Our goals were to: 1) quantify the prevalence of software in hazards, causes and controls; and 3) to evaluate software causes of hazardous conditions according to their specificity. We found that 49-70% of hazardous conditions in the three systems 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. Furthermore, 10-12% of all controls were software-based.

By analyzing hazard reports, we gained insight into risk areas within the software safety analysis process by analyzing its process artifacts. We identified six risks in software safety analysis reporting. 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.

In the future, we are planning to compare the various systems in an attempt to build baselines for the various software measures. This will allow us to interpret the data more effectively. For example, if the three systems are similar, then we might expect software to play a similar role in the causes and controls. If not, how might we characterize the differences? Analysis of the data shows that the software related hazards, causes, and controls for project B are much lower than those for project A. Why might this be true? The two systems may be sufficiently different with respect to their use of software, or the incompleteness of the data and the numerous transfers may be masking their similarities.

A longer term goal is to evaluate multi-system, "integrated" hazards. It may be difficult to consistently measure the software scope in hazard reports among subsystems because of difference in reporting software causes and controls (software's role), i.e., a sufficient software risk assessment across the program is difficult, expensive, or maybe even impossible. Finally, we plan to continue our evaluation to hazard reports on other NASA systems and extend our evaluations to include controls and verifications.

7 References

[Ariane5] ARIANE 5 Flight 501 Failure, Report by the Inquiry Board, Paris, July 19, 1996,. http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf.

[Basili84] V. Basili and D. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, vol.10(3), Nov. 1984, pp. 728-738.

[Basili08] V. Basili, F. Marotta, K. Dangle, L. Esker, and I. Rus, "Measures and Risk Indicators for Early Insights Into Software Safety," *Crosstalk*, Oct. 2008, pp. 4-8.

[Leveson93] N. G. Leveson and C. S. Turner, "An investigation of the *Therac-25* accidents," *IEEE Computer*, 26(7), July 1993, pp.18-41, doi:10.1109/MC.1993.274940.

[Prasad96] B. Prasad, Concurrent Engineering Fundamentals – Integrated Product and Process Organization, Prentice Hall, Upper Saddle River NJ, 1996.

Appendix A – "User guide" for specifying software causes of hazards

Documenting software causes and controls in hazard reports

As part of an OSMA SARP Safety Metrics Initiative, NASA SR&QA personnel and the Fraunhofer Center for Experimental Engineering have put together the following guide to assist safety engineers in **documenting software causes and controls of hazards in hazard reports for Phase I safety reviews**.

Four attributes must be specified for *each software cause and sub-cause* documented in hazard reports:

1. **Index the cause** – Label each software cause and sub-cause with a unique identifier.
2. **Identify the origin** – Indicate the CSCI that fails to perform its operation correctly.
3. **Specify the erratum** – Provide a description of the erroneous command, command sequence or failed operation of the CSCI.
4. **Specify the impact** – Provide a description of the erratum's effect that, if not controlled, results in the associated hazard. If known, identify the specific CSCI(s) or hardware subsystem(s) affected.

Attribute	Example acceptable values	Unacceptable values
Index	If cause 8 is "Software based control error," label sub-causes as 8.a, 8.b, 8.c, …	*{Software sub-causes not indexed}*
Origin	– Avionics CSCI – Propulsion CSCI – Vehicle Management CSCI – Timeline Management CSCI	– the software – the Flight Computer – a computer based control error – a general software fault
Erratum	– failure to detect a problem – failure to perform a function – performing a function at the wrong time – performing a function out of sequence – performing a function when the program is in the wrong state – performing the wrong function – performing a function incompletely – failure to "pass along" information or messages	– the software fails – the software is incorrectly designed – … other imprecise effects
Impact	– valve opens during incorrect segment – powering off hardware system X – system X placed in safe mode erroneously – cannot process messages from vehicle interface – allows battery cell voltage, current, or temperature to exceed limits	*{Impact not specified}*

When developing controls and verifications, each control must be traceable to each individual sub-cause, not with the set of sub-causes as a whole; each verification traceable to each individual control.

Attribute	Acceptable traceability	Unacceptable references
Control index	<Cause descriptions> Sub-cause 8.a … Sub-cause 8.b … <Controls> CTRL4 – The following address Sub-cause 8.a … CTRL5 – The following address sub-cause 8.b …	<Cause descriptions> Sub-cause 8.a … <text> Sub-cause 8.b … <text> <Controls> CTRL4 – The following controls address software-based errors …

<<Specific examples are provided on a second page but are redacted here for confidentiality reasons>>