Obtaining Valid Safety Data for Software Safety Measurement and Process Improvement

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

Lucas Layman, Kathleen Dangle, Madeline Diep
Fraunhofer CESE
College Park, MD
{llayman,kdangle,mdiep}@fc-md.umd.edu

ABSTRACT
We report on a preliminary case study to examine software safety risk in the early design phase of the NASA Constellation spaceflight program. Our goal is to provide NASA quality assurance managers with information regarding the ongoing state of software safety across the program. We examined 154 hazard reports created during the preliminary design phase of three major flight hardware systems within the Constellation program. Our purpose was two-fold: 1) to quantify the relative importance of software with respect to system safety; and 2) to identify potential risks due to incorrect application of the safety process, deficiencies in the safety process, or the lack of a defined process. One early outcome of this work was to show that there are structural deficiencies in collecting valid safety data that make software safety different from hardware safety. In our conclusions we present some of these deficiencies.

Categories and Subject Descriptors
D.2.8 Metrics: *Process metrics;* D.2.9 Management: *Software quality assurance*

General Terms
Measurement, Reliability, Experimentation.

Keywords
Case study, NASA, Risk analysis, Safety metrics.

1. INTRODUCTION
Software safety has become a prominent issue due to the larger role software plays in complex systems of systems. Contemporary systems are more distributed and network-connected than their predecessors and thus often contain greater amounts of software control. These systems are constructed using many suppliers whose products fulfill a wide variety of roles in the systems, all of which must be integrated into a uniform vision. Complex hardware and software systems, such as automobiles, defense systems and heart monitors, are often *safety-critical*. A *safety-critical* system is a system where a hardware, software or operational failure could result in loss of life, health or property. Assuring that such a system is acceptably safe is a challenging task that is often complicated by multiple suppliers using different approaches to address safety (and development in general).

The problem we address is how to evaluate software safety during the development of large, complex systems. Systems engineers, despite their expertise in hardware safety, are often unfamiliar with software risks. This inexperience is compounded when the software safety process is not well-defined, lacks prescriptive guidance, and is not uniformly applied. The challenge is to assist safety assurance personnel to monitor the quality of the software safety process, to identify areas of risk where software safety is not being appropriately addressed, to provide support for mitigating these risks and to recommend solutions for improving the software development process with regard to safety.

The goal of our case study was to implement a software safety measurement program on a large, safety-critical system of systems: the NASA Constellation program.[1] As NASA's next generation spaceflight program, Constellation consists of rocket propulsion, a crew module, operations, and many more systems. Our approach to measuring software safety, originally developed for a large U.S. Department of Defense program [2], provides early visibility into the implementation of the fault-tree based *hazard analysis process*. Our approach serves a two-fold purpose: 1) to assist safety assurance personnel in identifying system components of greatest software safety risk; 2) to identify potential risks due to incorrect application of the safety process, deficiencies in the safety process, or the lack of a defined process.

This case study describes the application of our software safety measurement program to three spaceflight hardware systems in the Constellation program. In this short paper, we focus on hidden process-related risks that we uncovered while applying our measurement program. These risks are the result of the inconsistent application of the software safety process.

2. BACKGROUND
Software has played a critical role in system safety for some time. The root failure of the Ariane 5 Flight 501 rocket explosion was a numeric overflow [1], the Mariner 1 satellite was lost during launch because the correct requirement was mistranslated and unverified [4], and race conditions in the software controlling the safety mechanisms of the Therac-25 allowed lethal doses of radiation to be delivered to four people [6]. While software generated the root causes for these failures, each failure was

ESEM'10, September 16–17, 2010, Bolzano-Bozen, Italy.

[1] Although it has been proposed that the Constellation program be terminated, several of the components of Constellation are expected to continue through implementation.

systemic in that the failure propagated through requirements, design, implementation, verification and project management.

Most research in software safety focuses on a model of risk where the software being developed contains critical flaws, and methods are developed to find and quantify those risks. Common safety analysis techniques, such as Failure Model Effects Analysis (FMEA) [7] and Fault Tree Analysis (FTA) [8], follow this model. These techniques identify various components of the system and their interactions, potential areas where there can be defects in the tree are identified, and the effects of those defects are traced through the software design. Corrections to mitigate those flaws are developed as modifications to the design.

2.1 The Software Safety Process on the Constellation Program

The hardware and software safety processes for the Constellation program are governed by a number of official process documents. These documents provide outlines for the safety analyses to be performed, when the analyses should be performed, on what system components, and what safety artifacts should be delivered for development milestones. The safety analysis techniques include hazard analysis, FMEA, probabilistic risk assessment, and many more. The process documents governing the use of these techniques often do not prescribe how software should be handled. In practice, the engineers must use their best judgment and consult NASA safety personnel on how best to apply these techniques to software. System safety is overseen by NASA safety assurance personnel, who evaluate the safety analysis artifacts for completeness and quality. Safety analysis artifacts, including hazard reports, FMEA worksheets, and system requirements that mitigate safety risks, are coordinated through a number of shared databases.

The focus of our software safety measurement approach has been the *hazard analysis process*. Hazard analysis is a fault-tree-based method that examines high level hazardous conditions (e.g. fuel tank explosion), identifies the potential causes of that condition, and creates design strategies or operational procedures for preventing those causes or mitigating their effects. The primary output of hazard analysis is the hazard report (HR), which documents the hazard and is stored in the Constellation hazard tracking system (HTS). Hazards have three attributes of interest:

- *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.

Project safety engineers with expertise in a specific area perform the hazard analysis. At various times, checkpoint meetings are held by the Constellation Safety & Engineering Review Panel (CSERP), which acts as gatekeeper for development milestones. 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 procedures or design strategies for mitigating these risks and makes recommendations for further analysis or the design of new controls. When applied to software, The CSERP assesses the hazard analysis process by measuring the levels of consistency and discipline that are applied in identifying and mitigating software-related hazards.

3. CASE STUDY DESIGN

Our goal is to develop and implement a set of process output measures that provide *visibility* into the system (and software) and the development process. These measures provide information that will make a lack of process visible, and provide insights into weakness in the current processes. The approach is based upon the premise that there is a relationship between the processes used during system development and the product's characteristics. A lack of process implies a risk of not achieving the anticipated product characteristics. Analyzing the execution of the process through intermediate process artifacts provides insight into whether appropriate processes are being performed and performed appropriately.

We first identified areas where software safety risk could potentially be measured by artifact analysis. We then performed the following steps:

1. Define **goals and questions** for each risk area to expose risks associated with software safety process artifacts.
2. Develop and enumerate **measures and interpretation models** based on threshold values.
3. Propose **responses** to identified risks, e.g., decisions and actions.

We examined the hazards of three Constellation flight hardware systems to assist NASA safety personnel in assessing software safety and to identify areas of potential risk in the software safety process. No measurement can guarantee software or system safety, but areas of risk can be identified. The three systems were in the preliminary design phase of development. Hazard analysis during the preliminary design phase focuses on clear and concise evaluation of hazard causes and the identification of an initial set of controls.

We address the following Goal-Question-Metrics (GQM) goals [3]:

1. Analyze the hazards reported for the three components 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 the hazard reports for the three components 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.

We used *specificity*, or how well specified were the hazards, as a measure of how thorough the development team was in performing appropriate safety analyses on their designs. If a hazard in the HTS used detailed information in specifying causes, we assumed the potential risk was better understood.

To evaluate specificity, we manually evaluated three attributes for each hazard cause:

- Origin – the name of the software component that fails to perform its operation correctly,
- Erratum – a description of the erroneous command, command sequence or failed operation of the software, and
- Impact – the effect of the erratum where failure to control results in the hazardous condition, and if known, the specific software or hardware subsystem(s) affected.

Each cause was given a rating of L1 (all attributes present), L2 (one or two attributes present), or L3 (no attributes present). The cause ratings were then used in an algorithm to generate an overall software cause specificity rating for the hazard ranging from La (highest, all L1 causes) to Le (lowest, all L3 causes). Additional information on these measures can be found in [5]

3.1 Data collection and analysis

We applied our analysis to 154 hazards containing 2013 causes and 4916 controls in the HTS for the three systems. We began by categorizing *cause* and *controls.* In the HTS, individual causes and controls are recorded as separate data fields. Each control is associated with a hazard and, optionally, with one or more causes in that hazard. A hazard may have 3-100+ controls, and 2-40+ causes. We identified each cause and control as software-related or not. A software-related cause or control described the behavior or design of FCSW (flight computer software), the FC (flight computer), a specific CSCI (computer software configuration items), or used the word "software." Controls could also be *transferred*, meaning that another hazard report defined the control. A cause was determined to be transferred when all of its corresponding controls were transferred.

The categorization data was recorded in a matrix that mapped controls (columns) to causes (rows). The same control could be applied to multiple causes, and causes typically had multiple controls. These matrices then served as the basis for counting a number of metrics used to answer questions for our GQM goals in the previous section. For full details of the data collection and analysis methods, including examples of the cause-control matrices, please see [5]. Please note that most of the analysis and counting was performed manually. Random sampling suggests a counting error rate of less than 5% for causes and less than 2% for controls.

4. RESULTS

Table 1 describes the prevalence of software within the HTS (Goal 1 for our study, given earlier). Across the three systems, 51% of the hazards contained at least one software cause (79 out of 154). What was also not apparent initially was that an additional 11 hazards (7%) that did not have a primary software cause had software as part of a control, which potentially can be the source of additional software hazards. That is, the failure of software as a control or monitoring mechanism can itself be a software cause of a hazard. Thus, close to 60% of all hazards were *software-related*. It is clear that software plays a significant role in the safety of the Constellation program.

Table 1. Software hazards in the HTS

	No software causes	**At least one software cause**
No software control	64 (42%)	0
At least one software control	11 (7%)	79 (51%)
Total	154	

A more revealing statistic is given by Table 2, where we look at the percent of hazard causes that were software. Of the total of 2013 causes, 601 (30%) were *transferred*. A transferred cause meant that another hazard report was responsible for documenting the controls for that cause. Of the remaining 1412 causes, only 204 (14%) were software causes. However, an additional 142 (10%) causes that were not caused by software issues had at least one software control. Thus software was involved in almost a quarter of all hazard causes, almost double the number of causes initially identified as software.

Table 2. Software causes in the HTS

	Non-software cause	**Software cause**
No software control	1066 (76%)	0
At least one software control	142 (10%)	204 (14%)
Transferred causes	601	
Total	2013	

Goal 2 for our study was harder to realize. Much like the earlier DoD study, traceability and specificity of the data was harder to quantify. Traceability and data integrity, at least with respect to software, were not as well defined as with hardware causes. Of the 79 hazards with a software cause, only 8 (10%) were highly rated for specificity (La), and 42 (53%) were poorly specified (Ld-Le). Part of the low specificity score was due to the three Constellation components being in preliminary design. Much like in our evaluation of the DoD project, we observe that additional process-related safety risk is introduced when safety engineers create and maintain the hazard data manually within the HTS. (Note: These systems are *not* inherently unsafe; only that the software quality assurance team cannot certify that due diligence was used on analyzing the system for safety risks.) In the next section, we describe weaknesses in the hazard process that allow for less precise measurements and hence more risk. A more complete analysis of the Constellation study is also available [5].

4.1 Software safety process risks

Our experience with an earlier DoD system [2] identified several problems in the implementation of a software safety process and hazard analysis in particular. Among these were:

1. **Software Hazard Identification.** Safety-related requirements are often not identified as such.
2. **Hazard Traceability.** The hazard tracking database does not provide sufficient linkages among the requirements documentation system, the test plan, and the defect tracking system.
3. **Data Integrity.** Hazards and their causes and controls may not be described in sufficient detail to be understood and verified.
4. **Level of Rigor.** There may be difficulty in differentiating among different levels of rigor (e.g., criticality of each hazard) for the various software safety requirements and identifying, assigning, and tracking the appropriate Level of Rigor (LoR) to specific software components that implement the safety-related requirement.

Much like our earlier DoD study, traceability of artifacts, requirements, and hazards is inconsistent for software. We identify several risks inherent in the software safety process that increase the risk of not providing sufficient insights into the process for appropriate quantification of software safety risk:

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. This has abated over time as CSERP has worked with safety engineers to form a uniform expectation of hazards.

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 projects or between hazard reports within the same project. A cause reading "Generic avionics failure or software flaw causes improper operation of control thruster" involves software but is not scoped to a specific software component as required by NASA procedure. This risk has abated over time due to experience, but is still present in some reports.

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

Many hazard reports place 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.

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. The HTS is being enhanced with some automated verification, traceability, and bookkeeping.

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

Many hazards have controls that contain enumerated "sub-controls." Greater confidence in the control may be gained by verifying the sub-controls independently. Controls are an explicit attribute of a hazard in the HTS, but sub-controls are not. Thus, references to sub-controls may become lost or incorrect as these references must be manually maintained.

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

Across the studied components, 30% of causes and 17% of controls were transferred. (Note: Controls were only available for two of the projects.) 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 hazard. Additional bookkeeping is necessary to ensure that the references to hazard reports, causes and controls are up to date.

5. CONCLUSIONS & FUTURE WORK

Many problems inherent in the hazard reports and the HTS are caused by an inadequate vision for the use of the HTS. The HTS is viewed as a storage repository by engineers, but in actuality is used to support analysis of the hazard reports by safety assurance personnel. It is important to make sure that (1) the HTS has adequate functionality, quality checks, and documentation; (2) there is traceability and synchronization among the various support systems (e.g., the HTS and the requirements management system and the defect tracking system); and (3) the quality of the data is monitored to minimize the need for data validation later on. The cost of not adhering to this advice is high rework costs and lower than desired system safety. Addressing these issues should be a part of the software safety development 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 one project are much lower than those for another. 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.

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 Software Reliability and Quality Assurance 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.

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. REFERENCE

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

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

[3] 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.

[4] P. E. Ceruzzi, *Beyond the Limits: Flight Enters the Computer Age*, MIT Press: Cambridge, MA, 1989,

[5] L. Layman, V. R. Basili, M. V. Zelkowitz, The Role and Quality of Software Safety in the NASA Constellation Program, Fraunhofer Center – Maryland Technical Report #10-101: http://www.fc-md.umd.edu/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), July 1993, pp.18-41, doi:10.1109/MC.1993.274940.

[7] R. R. Lutz and H-Y Shaw, "Applying Adaptive Safety Analysis Techniques," Proceedings of the 10th Int'l Symposium on Software Reliability Engineering (ISSRE '99), Boca Raton FL, pp. 42-49, November 1999.

[8] T. Maier, FMEA and FTA to Support Safe Design of Embedded Software in Safety-Critical Systems, CSR 12th Annual Workshop on Safety and Reliability of Software Based Systems, Bruges, Belgium, 1995.