A Methodology for Exposing Process Risk in Emergent System Properties

VICTOR R. BASILI, University of Maryland at College Park and the Fraunhofer Center for Experimental Software Engineering
LUCAS LAYMAN, Fraunhofer Center for Experimental Software Engineering
MARVIN V. ZELKOWTIZ, University of Maryland at College Park and the Fraunhofer Center for Experimental Software Engineering

Determining whether software and systems achieve desired emergent properties, such as safety, reliability, or security, requires an analysis of the system as a whole. This requires the system to be in the latter stages of development, when changes are difficult and costly to implement. In this paper, we propose the Process Risk Assessment (PRA) methodology for analyzing and evaluating such emergent properties earlier in the development cycle. Properties such as safety and reliability result from one or more development processes put in place to help achieve those properties. The PRA method analyzes artifacts from these processes (e.g., designs pertaining to reliability, or safety analysis reports) to determine: 1) whether the process itself is appropriate for achieving the desired property; and 2) whether the process is being followed appropriately. From PRA analysis, process risk can be quantified to indicate whether the system will have the desired properties. We applied this method to evaluate one emergent property, software safety, during the early stages of the development lifecycle for a network-centric, Department of Defense system-of-systems and several NASA spaceflight projects. We analyzed the safety processes implemented on these projects and their resulting artifacts. The PRA methodology identified potential risks in the software safety process and provided feedback to the projects for reducing these risks.

Categories and Subject Descriptors: D.**2.8 [Software-Software Engineering]**: Process Metrics; D.**2.9 [Software-Software Engineering]**: Management

General Terms: Management, Measurement

Additional Key Words and Phrases: Process risk, software safety, risk measurement

ACM Reference Format:

Basili, V. R., Layman, L., Zelkowitz, M. V., 20XX. A Methodology for Exposing Process Risk in Emergent System Properties. ACM Trans. on Soft. Eng. Method. X, X, XXXX.

DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Development of large systems in the aerospace, defense, energy, transportation and related industries is an expensive venture in terms of both time and money. One driver of the cost of these systems is that they often have challenging non-functional requirements, such as safety, reliability, security and performance. These *emergent properties* are particularly challenging to achieve because they evolve during development and can only be fully tested when the system is complete. Corrective actions based on the results of final tests or operations are often difficult and costly to implement. Nonetheless, failure to achieve these properties is costly and may be life-threatening at worst.

Achieving desired emergent properties is accomplished by applying specific processes that incorporate techniques for achieve the property, such as threat modeling for security or Failure Modes and Effects Analysis [Maier 1995, Lutz and Shaw 1999] for reliability. These processes must meet three, often unstated, assumptions:

Assumption 1. The process is capable of achieving the property and mitigating the risk of not achieving the property;

Assumption 2. The process is appropriate for the development context;

Assumption 3. The process is followed correctly.

If a process fails to meet any of these three assumptions, then there is a risk that the product will not achieve the desired property. The purpose of this paper is to define a risk measurement methodology for emergent system properties that does not focus simply on process conformance, but through its application enables the user to identify risks with the development processes and to create responses to identified risks throughout the lifecycle.

2. BACKGROUND

We define two types of risk with respect to emergent properties: product risk and process risk. A *product* or *technical risk* is the risk that a system will not achieve a desired property, such as functionality, reliability, performance, safety or security. For example, an incorrect software implementation is a technical risk that causes a rocket to prematurely activate its launch booster creates a safety risk. A *process risk* is the risk created by the (correct or incorrect) application of a process that leads to a product risk. For example, a software tester who writes poor performance tests contributes to the risk that the system may not meet performance requirements. All software development processes, including processes for mitigating product risk, introduce additional risks that must also be controlled.

Managing *product* risks associated with emergent properties is particularly challenging because these properties are a function of the system as a whole. It is difficult, if not impossible, to test these properties with a high level of confidence before the system is completed. Furthermore, methods for testing properties such as security, safety and performance are often ill-defined, immature and non-repeatable. Economically, it is much more advantageous to address risks early in the development cycle when required changes are easier and less costly to make, particularly in large, complex systems.

Mitigating *process* risks often takes the form of quality assurance, process conformance evaluations, or the application of process improvement frameworks such as CMMi [Chrissis et al. 2003]. Identifying the cause of process risk is where most projects struggle. The first instinct is to say "you're not following the process" or "you're not doing enough of the process" (see *Assumption 3,* Section 1), and, indeed, research has shown that teams who follow process consistently tend to perform better [Krishnan and Kellner 1999]. But research has also shown that "even companies that use the same development process at the same level of maturity don't achieve the same levels of quality" [Pardo et al. 2011]. While not following a process can lead to risk, there are often hidden reasons that correlate with process nonconformance – the process is flawed or the context does not allow various steps to be performed (see *Assumptions 1* and *2).* For example, there may be conflicting goals, such as when developers abandon process when faced with an imminent deadline. Perhaps an organization's performance testing guidelines are simply ill-specified and uninformative, and thus the performance testing process is ad-hoc and ineffective. In practice, pressing a developer to apply more process may not be the solution; there is often an underlying cause for process nonconformance.

2.1 Leveraging Process Artifacts to Identify Process Risk

Our approach to mitigating those risks associated with emergent properties focuses on *process risk management*. In particular, we focus on the *process artifacts* that capture information about emergent properties. For example, design documents can provide insight into reliability, and fault tree analyses as part of hazard analysis [Dehlinger and Lutz 2004] contains information relevant to system safety.

The artifacts of a development process contain two types of information that provide insight into the process: syntactic information and semantic information. An

artifact's *syntactic information* is the data elements and their expected compositional format. For example, the syntactic information of a defect report may include the description of the defect, the reproduction steps, and a criticality of the defect. An artifact's *semantic information* is the meaning or interpretation of the syntactic data in the context of development. Interpreting semantic information almost always requires human expertise. For example, a computer cannot automatically infer the risks and necessary mitigating actions required based on the contents of a defect report.

We assert that potential product risk increases when the ability of an artifact's consumer (e.g. developer, project manager, tester) to perform a semantic analysis (e.g. to determine if the project is on schedule, to understand the bug) is compromised by syntactic problems. The process artifacts are often *the only evidence* available during development that provide evidence that steps have been take to mitigate product risks and that these steps have been applied appropriately. Insufficient, incorrect or missing syntactic information in process artifacts is a strong impediment to accurate, useful semantic analysis and is the indicator that we leverage to identify and measure potential risk during the development process.

Our process risk measurement methodology enables users to answer the first three questions and provides insights into the fourth question about processes and process artifacts:

1. Do we have any information in the process artifacts?
2. Do we have enough information to perform a syntactic analysis of the data?
3. Do we have enough information to perform some form of semantic analysis of the data?
4. Is the data semantically correct?

A positive answer to each successive question provides greater insights into the development process, with a correspondingly deeper understanding of the risks that may be present.

3. THE PROCESS RISK ASSESSMENT METHODOLOGY

In this section, we introduce our risk measurement approach, called the Process Risk Assessment (PRA) methodology. The PRA method is comprised of six steps, which are applied iteratively to form a risk measurement plan. We explain each step in detail and provide an example of applying the methodology from the NASA Constellation program to evaluate software safety risk. The six steps are grouped into three stages, as follows:

I.	Identifying measureable insight opportunities	1.	Identify **insight areas** (e.g., process artifacts) from the development process that provide insight into risk areas.
		2.	Identify **measurement opportunities** that provide insight into each insight area.
II.	Evaluating the quality of information	3.	Develop **readiness assessment questions** to identify if sufficient information exists to implement process risk measures.
III.	Measuring, interpreting, and providing advice	4.	Define **goals, questions, and measures** for each risk area to expose risks associated with process artifacts.
		5.	Develop and enumerate **models** of how the measures will be **interpreted** via threshold values.
		6.	Propose **responses** to identified risks (e.g., decisions and actions) in order to mitigate those risks.

The first three steps of PRA determine if process risk analysis has the potential to yield meaningful information. If insight areas or measurement opportunities cannot be identified (Steps 1-2) and the artifact's syntactic and semantic information cannot be readily assessed (Step 3), then the development process itself may be a source of

risk. The final three steps of PRA define process risk measurements, develop interpretations of the measurement results, and establish mitigations for identified risks. For the practitioner, we have defined "keys" to each of the six steps in a table format (See Appendix B).

3.1 Constellation Case Study

To illustrate the PRA method, we provide examples from applying PRA to identify potential software safety risks within the NASA Constellation program[1]. The Constellation program was a complex system of systems to facilitate the next generation of human spaceflight. We examined three large spaceflight systems in the preliminary design stage that contained numerous hardware and software *safety* features. Safety is an ideal example of an emergent product property that cannot be fully tested until the system is operational. A safety risk is a *hazard*, which 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 [DoD 2000]. Hazards caused or contributed to by software have become a greater concern in systems development as many traditionally hardware-centric systems become highly reliant on software. Catastrophic safety failures due to software are well-documented in the literature, e.g. [Nuseibeh 1997, Leveson and Turner 1993]. On Constellation, software was safety-critical in numerous areas from propulsion and navigation to maintaining a livable environment inside the crew capsule.

We partnered with the software Safety, Reliability and Quality Assurance (SR&QA) group for Constellation to gain insights into potential software safety risks associated with the software safety process. In particular, we focused on the *hazard analysis process*, which is an analysis conducted during the design phase to help identify causes of hazardous safety conditions and to develop mechanisms to avoid, control, or otherwise mitigate those causes[2] [FAA 2008]. The official Constellation hazard analysis process, *CxP 70038 – Constellation Program Hazard Analyses Methodology*, prescribes the "methodologies and processes required […] to implement a healthy and robust hazard analysis, hazard communication, and hazard approval process" [NASA 2009]. CxP 70038 required that software be analyzed for safety-critical procedures that could lead to a potential hazard. Our goals were to help SR&QA evaluate software-related hazards, to help software quality engineers implement software hazard analysis processes, and to demonstrate conformance to the hazard analysis processes with respect to software.

Throughout the development of Constellation, project safety engineers identified potential system hazards and created *controls* (i.e. strategies) for mitigating those risks. Hazards were recorded in a program-wide Hazard Tracking System (HTS). At each program milestone review, the development groups presented their hazard analyses to the Constellation Safety & Engineering Review Panel (CSERP), who reviewed the hazards and evaluated the control strategies, acting as gatekeeper for development milestones.

SR&QA management wanted to understand where in the system software played a role in possible hazards, either by causing a hazard or by controlling it. By understanding the role of software in system hazards, SR&QA could identify systems and subsystems with the greatest potential software safety risk and take appropriate management actions. The results of the Constellation case study are more fully documented elsewhere [Layman et al. 2011].

---

[1] Although the Constellation program was cancelled, some of the projects are expected to be continued. Our results are being applied to other NASA projects as well.

[2] See Appendix A for an overview of hazard analysis.

3.2 Step 1: Identify Insight Areas from the Development Processes that Provide Insight into Risk Areas

The first step of the PRA method is to identify intermediate outputs of processes. These artifacts can provide insights into process conformance and effectiveness in achieving the desired emergent properties. We ask, what potential syntactic and semantic information can be gathered from process artifacts to provide risk insight? If no artifacts exist that provide insight into risk, then it is likely that risk will be present in the system because there is no process or artifact capturing that risk. The first step identifies insight areas at a very high level as candidates for further investigation.

Constellation example

The Constellation SR&QA manager identified the hazard analysis process as a readily-accessible source of software safety information early in the development process. Hazard reports are the intermediate output of the hazard analysis process. All hazards, including software-related hazards, are stored as *hazard reports* in the Constellation hazard tracking system (HTS). These hazard reports contained the syntactic and semantic safety information that provided insight into potential software safety risks. Based these reports, we identified the following potential insight areas:

- *The set of hazards, with its causes, controls, and verifications.* The hazard reports can provide insight whether the program is adequately identifying and documenting the required software safety information;
- *The relationship between hazard causes, controls, and verifications.* These relations provide insight into whether sufficient actions are taken over time, i.e. that the hazard controls are being implemented and verified.

3.3 Step 2: Identify the Measurement Opportunities that Provide Insight into Each Risk Area

Second, each insight area is evaluated for *measureable* indicators of process risk. A measurement opportunity can measure process conformance (e.g. are all of the fields in the test results filled in?) or the software risk directly (the number of defects found during test). A measurement opportunity is identified by creating a *high-level, informal metric* based on the information available. This step identifies (and thereby excludes) processes or process artifacts that do *not* have measureable information. A process that does not have measureable output may represent a risk itself, and is an indicator that the process or process artifacts do not contain meaningful information.

Constellation example

We identified measurement opportunities to help quantify software safety risk based on *information* captured in the hazard reports. Because we were analyzing hazard reports in preliminary design, only the hazard causes had to be well-described according to CxP 70038. The hazard analysis process, together with other NASA standards, described what constituted a "meaningful" description of a hazard cause. Thus, we identified measurement opportunities that focused on syntactic information to help evaluate conformance to the prescribed hazard analysis process, but which were also related to the quality of the semantic information (i.e. "meaningful" information). Other measurement opportunities were derived from additional requirements in CxP 70038.

Those measurement opportunities were:

- Evaluate hazard causes, controls and verifications to determine if they contain the required syntactic components that are prerequisite for a "meaningful" hazard analysis.

- Count the number of hazards, causes, controls, and verifications that involve software to quantify software involvement in hazardous conditions.
- Verify that each cause is addressed by at least one control and that each control is addressed by at least one verification.
- Count the number of causes, controls and verifications that are "transfers". Transfers are a reference to a cause, control or verification in another hazard report (e.g., another hazard report addresses appropriate mitigation of this risk). All transfers must be traced and verified as completed for a hazard report to be considered "closed."

In these first two steps, we have not gathered any actual data; we are only looking at the process outputs and their potential to be measured. The purpose of these steps is to understand the development process we wish to measure (e.g. the software safety process) and to understand the process artifacts. These steps may seem superfluous, but are necessary. Too often we assume that a process is producing meaningful, insightful information, but this information may not be, in fact, available or useful.

3.4 Step 3: Develop Readiness Assessment Questions to Provide Risk Status and Identify If the Insight Area Can Be Evaluated

In the third step, we determine whether the syntactic and semantic information in the process artifacts contains sufficient information to investigate the risk further. We propose a set of *readiness assessment questions* that allow us to gain additional insight into the areas of interest, to get a quick and easy status report of the area, and to identify whether it is possible to go deeper. What is learned from these questions helps tailor the models, measures, and responses applied at a deeper level. For example, we might ask "are software safety-related requirements specially noted in the requirements repository?" If not, there may not be sufficient information to evaluate software safety risks. Answering these readiness assessment questions is an iterative process, as answering one question may lead to others. Processes and process artifacts that do not pass a readiness assessment are indicators that the process is not being followed, or that the processes is not well-defined for the current context. In both cases, the project should take steps to correct the development process.

Constellation examples

In the Constellation case study, we developed many readiness assessment questions while exploring the measurement opportunities. Examples of these questions include:

- ***Can we access the hazard tracking system and hazard reports?*** Access to the process artifacts was a necessary precondition to any measurement. The HTS was made available to us, but the HTS contained hazard reports for only one of the three spaceflight hardware project on Constellation that we examined (see next list item).
- ***Is the content of the hazard tracking system up to date?*** The hazard tracking system only provided up-to-date hazard reports for one of the three spaceflight systems (Project A). Project B's hazard reports were in the HTS but not visible as the development contractor 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. However, we were able to obtaining the hazard reports for Projects B and C from a Wiki containing CSERP review meeting materials.

- ***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 potential failure causes and to develop preliminary controls. As this was still early in the Constellation program, verifications were not yet specified nor required by CxP 70038. As such, we could not measure them. 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.

In addressing the readiness assessment questions, our first observation, as stated above, was that few of the hazards had been entered into the HTS by safety engineers – we had to obtain the hazard reports from the CSERP review materials. However, keeping the HTS up to date is an easy first step in mitigating hazard risks and increasing the effectiveness of the hazard analysis process. The HTS has the capability to track "transfers" in hazard reports automatically, making traceability maintenance a much lower cost than the manual effort required of engineers working only with word processor documents. Furthermore, the contractor withholding intermediate hazard information from the HTS prevented SR&QA management from easily assessing the progress of the hazard analysis process.

3.5 Step 4: Define Goals, Questions and Metrics for Each Risk Area to Expose Risks Associated with Process Artifacts

Having determined in steps 1 to 3 that we have sufficient information to make deeper risk assessments, in the fourth step we set goals for identifying specific software risk based on the syntactic and semantic information available in the development process artifacts. We develop a set of questions that will help determine if those goals are being met, and we identify formally the metrics we will use. This step is an application of deriving goals, questions, and metrics in the GQM approach [Basili and Weiss 1984]. Goals and questions should focus on the application of the development process and the expected information in the process artifacts. The premise of our methodology is that information in process artifacts should meet the expectations of the processes, and, if not, there is risk that the process is not appropriate for achieving its goals or that the process is not being applied appropriately.

GQM questions can be asked about process risk, such as "is all of the information in a software safety requirement recorded?" We can also ask if the content of these safety requirements is semantically meaningful. Goals and questions can be asked from a more product-oriented perspective as well, such as "which parts of my system have the most software safety risks?" By answering these questions with metrics, we enable quantifiable comparison, measurement repeatability, a baseline for measuring future changes, and well-defined targets for future efforts.

Constellation examples

SR&QA identified several goals for software safety risk analysis based on the hazard reports. After applying PRA steps 1-3, we worked with SR&QA to specify two goals for analysis.

By understanding the role of software in system hazards, SR&QA could identify systems and subsystems with the greatest potential software safety risk. To this end, we created the following GQM approach to characterize software's prevalence in hazards:

Table 1. Goal 1 - Prevalence of Software

Goal 1 - Prevalence of software Analyze the available set of 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.
Questions Q1-1. What percentage of hazard causes are software causes? Q1-2. What percentage of the hazards is software-related? A software-related hazard has at least one software cause or software control. Q1-3. What percentage of hazard causes have software controls? Q1-4. 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. Thus, this software control can again be the cause of a hazardous condition.
Metrics M1-1. The total number of hazards, causes and controls M1-2. The number and percentage of software hazards M1-3. The number and percentage of software-related hazards M1-4. The number and percentage of software causes M1-5. The number and percentage of software controls.

Given SR&QA's second goal of identifying the systems and subsystems with the most risk, we identified several subgoals, one of which was to evaluate whether the syntactic and semantic information in software causes of hazards adhered to established NASA guidelines. If software cause descriptions do not syntactically conform to the information required by process standards, then there is a risk that not enough semantic information is captured to adequately describe the hazard cause.

Table 2. Specificity of Software Causes

Goal 2 - Specificity of software causes: 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.
Questions Q2-1. What number and percentage of software causes is *well-specified* according to the Constellation hazard analysis methodology requirements? Q2-2. What number and percentage of software causes is *partially-specified?* These causes lack certain pieces of information needed to evaluate their quality. Q2-3. What is the number and percentage of software causes is *generically-defined?* A "generic" cause (e.g. "the software fails") is not specific enough to identify any control strategy.

Metrics
M2-1. For each hazard report, count the number of *L1* causes
M2-2. For each hazard report, count the number of *L2* causes
M2-3. For each hazard report, count the number of *L3* causes
 	
• L1: a specific software cause or sub-cause[3] for a hazard, which *must* include all of the following:
  ○ Origin – the CSCI (e.g., software component) 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 which 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, which 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, which does not specify the origin, erratum or receiver at the CSCI/hardware subsystem level.

[3] 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.

While Goal 1 and Goal 2 both deal with syntactic information in the hazard reports, Goal 2 begins to bridge the gap between complete syntactic information and meaningful semantic information. The syntax for a "well-specified" hazard cause (as inferred from NASA standards) enables meaningful semantic information (though does not guarantee it).

3.6 Step 5: Develop Interpretation Models and Threshold Values of Measures

Using the metrics developed in Step 4, we need to interpret the data. What are good values and which values represent risk? For example, for Goal 2 above, what percentage of "generically-defined" (L3) software causes is too high, and what are the implications for the project? Ideally the interpretive models are based upon thresholds established from prior projects. If no such data exists, experts can provide proxies for the thresholds. One side effect of the PRA method is that the collected data can be used to create baselines that can be used as thresholds in future projects.

Constellation examples

We analyzed a total of 154 hazard reports for the three Constellation systems: 77 in Project A, 57 in Project B, and 20 in Project C. The analysis of each hazard report was performed manually by reading the text of the causes and controls. In total, over 2000 causes containing nearly 5000 controls were examined.

Goal 1 did not have an interpretation model, per se, as it was meant to provide a description of software's role in hazards to provide stakeholders with an understanding of the importance of software safety. Table 3 provides the statistics that answer the questions in Goal 1.

Table 3. Measuring the prevalence of software

	Question	Project A	Project B	Project C
Q1-1	What percentage of the hazard causes are software causes?	15%	12%	17%
Q1-2	What percentage of the hazards is software-related?	49%	67%	70%
Q1-3	What percentage of the causes has software controls?	29%	23%	-
Q1-4	What percentage of hazard causes are hardware causes with software controls?	14%	11%	-

The analysis began with finding syntactic keywords (e.g. "software", "flight computer") that gave an indication that a cause or control was software-related. We found that 49% of Project A's hazard reports, 67% of Project B's hazard reports, and 70% of Project C's hazard reports were software-related. This indicates that software is a safety-critical aspect of the overall system and over half of all hazard reports are software-related. The importance of software clearly demonstrates the need for a strong software development process with adequate control and verification.

Goal 2 provided insight into the execution of the hazard analysis process itself. In this example, we create an interpretation model for the question "what percentage of software causes is *generically-defined*? (Q2-3, M2-3).

> **Interpretation model:** ***if*** *L3 > 0* ***then*** additional work is required to better specify those software causes and those causes must be-revaluated for specificity at a later time; ***otherwise*** the causes are ready to be evaluated for quality by domain experts.

A "generically-defined" cause is not specific enough to specify a design feature or operational procedure that could function as a control strategy. Thus, all generically-defined causes should be improved to be better specified. In the three Constellation projects, Project A had 38 L3 causes (29%), Project B had 37 L3 causes (22%), and Project C had 16 L3 causes (29%). None of the projects met the criteria of the interpretation model.

At its core, Goal 2 examines process conformance by looking for syntactic information in the cause description. Many causes did not follow the prescribed syntax, and the resulting semantic information was insufficient for identifying controls. As discussed early in the paper, the initial reaction may be to say "you're not following the process correctly." However, SR&QA personnel acknowledge that non-conformance in this case was not necessarily a failure on the part of the safety engineers; the integration of software safety with traditionally hardware-centric system safety processes (such as hazard analysis) was still a work in progress. The guidelines for specifying software causes were scattered over five different standards and process documents, and each project reported and scoped software causes differently. In this case, the lack of process conformance was partially attributed to needing process guidance for the current context.

3.7 Step 6: Propose Responses to Identified Risks

The goal of this step is to propose actions to be taken in response to any identified risks. If the interpretation models indicate that the process is not being followed and/or not producing the information expected, stakeholders must endeavor to understand why the process is not being followed. Again, this may be more than a compliance issue, and could result from inapplicability of the process to the current context or an ill-defined or not sufficiently defined process. In such a case, additional training may be necessary for the developers, the process may need to be refined with input from the practitioners, or entirely new techniques may need to be applied that are more suitable to the context. Responses to identified risks may also focus on the product, such as increasing the amount of testing for high risk components or requiring additional review of proposed architectures.

Not identifying any risks at this step does not mean that there are no software-related risks. It only means that the development team seems to be following the process in a reasonable manner. Specific risks may not have been identified because measurable process artifacts were not present. Furthermore, technical risks (i.e. the system may not operate as expected) may still be present in the details of the design or implementation that is being worked on.

Constellation examples

SR&QA integrated the percentage of software-related hazards (along with other measures related to Goal 1) into a risk scorecard of the systems and subsystems with greatest potential software safety risk. This scorecard can help manage safety support effort. The data related to Goal 1 were used to inform other members of Constellation program management as to the importance of software in overall system safety. These metrics results were surprising to many. The risk scorecard was to be used to track the evolution and improvement of software safety risk over the course of the project. Unfortunately, the program was canceled shortly after this analysis, and thus we could not observe how the data would be used as part of proactive program management. The following response is a notional example of what project responses might look like:

- Allocate the available work hours for software safety assurance according to the rank order of subsystems with the most software-related hazards.

The responses to risks evaluated in Goal 2 focused on improving the quality of the software cause descriptions. As described in the previous section, the process risks associated with Goal 2 were largely attributed to project safety personnel being unfamiliar with the hazard analysis process and how to incorporate software safety into that process. As such, the responses to these risks focused on process improvement and providing better support in both training and in the HTS for reporting software causes. Notional example responses to Goal 2 include:

- If the software cause does not specify a specific origin (i.e. "the software fails"), then assist the project safety engineer to differentiate between the architectural components of the software (e.g. the propulsion control system, the avionics component).
- If a hazard report contains L2 or L3 software causes, have the project safety engineer who authored those causes rewrite them using the "User Guide for Specifying Software Causes" we produced with SR&QA as guidance and reevaluate at the next Technical Interchange Meeting.

4. UNCOVERING PROCESS RISKS USING THE PRA METHOD – CASE STUDY RESULTS AND LESSONS LEARNED

We have applied the PRA method to evaluate software safety on three projects to date: 1) the Constellation study in the previous section; 2) a large, network-centric US Department of Defense system-of-systems; and 3) a NASA-developed satellite. Descriptions of the DoD and NASA satellite systems are provided in Appendix C. In all three case studies, the programs relied on hazard analysis to describe, analyze, and reduce software safety risks. By applying the PRA method, we uncovered a number of process risks in each project with respect to the application of hazard analysis to software safety[4]. In this section, we identify three process risk "themes" that were common across the projects and thus provides guidance to other programs implementing software safety analyses to help avoid these risks.

1. <u>The inability to track software hazards and software safety requirements</u> against the backdrop of system hazards and system safety requirements. On Constellation and in the DoD project, software safety risks were often not specifically marked or headlined in the hazard reports. Identifying which hazard reports described software was time-consuming: different hazard report authors had different writing styles and sometimes different terminology, and the hazard reports themselves could be significantly lengthy[5]. Software safety management personnel were forced to manually collate and track this information, which became an expensive, but necessary, requirement for proactive software safety management across these large projects.

On all three programs, many hazard controls were not identified as software-related even though the controlling mechanism included software, and the corresponding software requirements for implementing the control were not marked "safety-related" as required. The inability to easily identify software causes and controls and the non-uniformity of the syntactic content in the hazard reports also became apparent when applying Steps 4 of the PRA. As we mentioned, the PRA method is iterative, and Goal 2 in the Constellation study was created in response to this observation.

---

[4] The specific process risks for the DoD and Constellation case studies are described in Basili et al. [2008] and Layman et al. [2011] respectively.

[5] The Constellation hazard reports, for example, had a median length of 36.5 pages (min: 8, max: 152, mean: 43.2).

The safety analysis processes did not adequately distinguish between software and non-software safety concerns. This forced software safety management to manually collate and track this information, which became an expensive, but necessary, requirement for proactive software safety management across these large projects. Tracking software safety concerns would be easily enabled if safety requirements, hazards, and other safety analysis artifacts contained meta-data (e.g. a checkbox) indicating whether or not those items were software related.

The differences between software and software related hazards were not handled adequately. In the hardware realm, if a hardware control was specified, then there would be a control to ensure that the original hardware control did not introduce a new hazard. This was evident in the hazard reports of the NASA satellite. But for software controls to hardware causes, no such control was included to ensure that the added software did not introduce new risk.

2. Inadequate traceability exists between safety requirements, hazards, causes and controls. In all programs, bi-directional traceability is required between safety requirements, associated hazards, causes, controls and verifications to ensure that safety requirements have been implemented. In all programs, bi-directional traceability was not present in the artifacts we examined. Retroactively inserting this traceability as the project nears completion (as is the historical practice) is expensive. In the Constellation and DoD programs, the hazard reports contained sections for safety requirement references, though these sections were not filled in or were marked as "To Be Determined" (TBD). The NASA satellite program contained some references to safety requirements, but the hazards were not completely filled in or were marked as "TBD".

On Constellation, we observed a number of hazard reports where the internal references to other hazard reports' causes and controls (i.e. "transfers") were missing or incorrect. Even when the references were correct and up-to-date, significant manual effort was required to build the dependencies between causes and controls across hazard reports. Across the three Constellation systems, 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 because they inhibit traceability and require more effort to understand and maintain. On the NASA satellite program, an external document was used to keep track of software causes and controls across hazard reports, somewhat lessening the traceability burden.

3. Inconsistent scope and unstructured details when describing hazards, causes and controls. In both programs, safety engineers wrote their hazards, causes and controls in unique ways. This made consistent evaluation on the part of safety management difficult. For example, on Constellation, the content of the hazard reports differed substantially between the three spaceflight systems and among hazard report authors within the same program. All three of the spaceflight systems approached software hazard analysis differently – some hazard reports described all software issues under one large cause and control, while others spread software issues throughout multiple causes.

For example, 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. Furthermore, inconsistent structure and vocabulary precluded an automated syntactic analysis of the safety artifacts. This was not simply a matter of non-conformance to the safety analysis processes, but was the result of different interpretations of processes that were not concretely defined. 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. A constant challenge faced by safety engineers is appropriately separating complex hardware and software functionalities into multiple causes and controls. 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

These themes may be indicative of large engineering projects being developed by many organizations. Our observations indicate that simply defining a development process is not sufficient to identify safety (or any other kind of) emergent risk. Management, measurement, and feedback of the process actually being used is as important as defining a proper process in the first place.

4.1 Lessons Learned for Implementing Software Safety Analysis Processes on Future Programs

All of these programs were attempting to integrate software safety with traditional safety processes that originated in hardware and system reliability. For all programs, elevating software safety to a level of importance equivalent to hardware and system safety was challenging. Defining how software should be incorporated into traditionally hardware-oriented analyses (such as hazard analysis) is still very much a work in progress. In these traditional hardware-oriented environments, software is often viewed as just another black box hardware component. However, many high-profile system safety disasters (e.g. Ariane 5 [Nuseibeh 1997], THERAC-25 [Leveson and Turner 1993]) can be attributed to defects in the software and software processes that propagated throughout the system. The analysis, detection, and mitigation of software risks are not isolated to the software, but involve the entire system engineering process.

Software risk analysis is not the same as hardware risk analysis. Software failure rates are difficult to predict, and mitigation strategies for hardware failures (e.g. redundancy) often do not apply to software. Furthermore, software controls are difficult to specify because software properties and constraints are not well understood early on. The Ariane 5 disaster [Nusebeih 1997] is an example where a runtime control (e.g. testing for overflow) was intentionally disabled to meet performance requirements. Software safety risk management should not be isolated, however. The interaction of hardware and software is a source of risk, and software failures will almost always manifest in some loss of system function. Software safety risk management must be integrated in the overarching system safety process in such a way to account for the unique ways that software risk must be managed, while recognizing that software safety risk is of equal importance in the overarching system safety process.

From our results of applying the PRA method on the three programs, we have identified the following lessons learned for future software safety-critical programs.

1. <u>Provide explicit guidance for applying safety analyses to software.</u> Assuring software safety is not as simple as saying "apply the hazard analysis process to software". As we observed in the Constellation case study in particular, a lack of consistent understanding of how to apply hazard analysis to software led to both over specified and underspecified software cause descriptions. The CSERP review was not interested in reading about state-transitions in software that would undoubtedly change, nor was a description of "the software fails" good enough. The safety engineers often spent several iterations simply getting the *description* of the

cause to be acceptable before an analysis of the actual design features (i.e. the controls) could even begin. Providing guidance to the safety engineers on how to specify software causes saves time and effort on the part of the engineers because it reduces the number of iterations required to get the syntax correct and the semantics meaningful. For the Constellation program, we developed a two-page guideline for the various development teams to use in filling out hazard reports. Although current NASA guidelines give the contents expected in hazard reports, they do not give a clear indication of the level of detail or the format of this information. Our guideline, when followed, provides consistency across development teams, which allows CSERP to more readily monitor process risks across multiple developments.

2. Plan for automated analysis and traceability and promote usage of the HTS capabilities. In our studies, we observed inadequate planning in the use of the hazard tracking systems, which impaired the ability to track and measure software safety risks. Ostensibly, the hazard tracking systems were designed: 1) to capture the hazards in a consistent format; 2) to house the syntactic data of the hazards to enable rudimentary searching, filtering and analysis. The value of the HTS as a management tracking system is further increased by providing automatic support for traceability between hazards and hazard attributes. In the NASA satellite program, no hazard tracking system was used. In the Constellation and DoD programs, it was clear that goals for searching, filtering and analyzing hazard reports to support risk management were not considered. For example, there was no way to identify software-related hazards among the set of system hazards, despite the fact that software safety management was handled by special groups in each program. Several hundred person-hours were spent by us to analyze the Constellation hazard causes and controls to identify merely if software was involved. However, as we were able to demonstrate in a prototype HTS, adding a checkbox next to each cause or control denoting "this is software-related" enabled an automated search for software causes and controls that took mere seconds instead of hundreds of hours. And even though there was such a box in the DoD system for hazards in general, software-related hazards were not identified properly even when it was clear that software was the cause or control of the hazard.

In Constellation and DoD, the traceability features of the HTS were not used by the safety engineers, who preferred to author the hazard reports in a word processor and then copy and paste in the HTS. The HTS could provide a number of useful features regarding traceability, such as verifying that references and links are still valid, automatically updating traceability links, and detecting dependencies. Leveraging these capabilities would avoid process risk and unnecessary effort.

3. Require software safety management and measurement in the acquisition process. The vendor acquisition process must promote the importance and iterative measurement of software safety. On programs as large as Constellation and the DoD system, the development effort can include dozens of vendors over several years. The sources of information that can be used to measure software safety are limited to the process artifacts that are provided by these vendors. In the Constellation program, for example, the hazard reports were only required to be available for milestone reviews. Such milestones could occur months or even years apart, and were not necessarily available on demand per the contract. As such, NASA's oversight and direct involvement in the software safety assurance process was somewhat limited. As another example, if one wishes to track defects over time, then visibility must be provided by the vendor into their defect tracking database (or some proxy). If this is not written into the contract, then there is a risk that such information will be unattainable and the associated risks not measurable.

5. CONCLUSION

We have seen that the PRA method can uncover potential process risks and provides a means for evaluating emergent properties early in the development cycle. We have learned several lessons and observed some limitations in applying the PRA method over the course of these three studies.

5.1 Lessons Learned about PRA

The PRA method is both an evaluative method of a process, and a method to improve the process while it is being applied. PRA is most effective as a quality assurance activity where the methodology is applied by people familiar with but outside of the process. The first assumption of applying the PRA method is that there is a process, whether it is implicit or explicit, that can be analyzed. Ad-hoc methods of achieving a desired product property are not candidates for the PRA method of detecting process risk.

If the process you wish to study is implicit, making it explicit can reveal undefined "grey" areas, differences in interpretation, etc. In both our Constellation and DoD studies, we received valuable insight from software safety personnel involved in program management, but were not performing the safety analysis itself except in a review capacity. From these individuals, we obtained insight into the goals of the safety process and insights into applying those processes in their respective contexts that were invaluable in interpreting the risks we observed. Expert domain knowledge is necessary to interpret the semantic meaning of technical artifacts, but the risks we uncovered in the processes did not require significant technical expertise but a thorough understanding of the process requirements and goals. However, interaction with the experts is critical to validate goals, to verify findings, and to create willingness to adopt proposed responses to risks.

The PRA methodology is iterative. For a given application, readiness assessment questions may not pass, thus requiring you to start over in identifying insight areas. The metrics you interpret may raise further questions (e.g. after counting the number of software defects, what is really needed is how severe they are), causing you to reformulate goals and questions. It is very likely that the process artifacts that you plan to examine (e.g. safety requirements) do not contain the information you desire or the quality of information is poor (both of which are indicators of process risk), and thus you will need to rethink potential insight areas while also looking at how to improve the process.

Step 3, asking Readiness Assessment Questions, although a simple step, found significant risk in the projects we studied. This step is crucial to challenging the assumptions of most risk models (as described in Sections 1-2). We believe, from anecdotal evidence, that these assumptions do not hold true across a large number of software developments, and are the sources of significant development risk. Step 3 should always be completed using actual project data before proceeding to steps 4-6. Step 3 may cause you to reevaluate steps 1 and 2 regarding where you can look for data and what's important. In the DoD study, we could have saved significant effort since the data required for steps 4-6 was simply not available. For Constellation, asking readiness assessment questions resulted in changing the method for obtaining the hazard reports we needed to evaluate.

Models and interpretations help shape goals. Goals, measures, models will vary according to when the measurement takes place as the process can have different expectations/outputs at different points in time. For example, all the projects we studied were in the preliminary design phase when only requirements and high-level designs were available for safety analysis. The responses to the identified process risks were to improve the process and provide process support, and the response to product risk is to change the design of the system. At a later stage of development,

say testing and verification, one could obtain more concrete metrics on the state of system safety with respect to actual system behavior. At that point, the responses are more limited in that changing design and implementation will be extraordinarily expensive (especially in a large system).

5.2 Summary

The PRA method helps address and respond to software development process risk in achieving emergent system properties, such as reliability, safety and security. Our six-step method does this by challenging the following assumptions in the application of development processes:

- The process is an effective way of achieving the property and of mitigating the risk of not achieving the property;
- The process is appropriate for the development context;
- The process is followed correctly.

The PRA method provides visibility into process risks throughout the development lifecycle by measuring process conformance through the analysis of syntactic and, possibly, semantic information contained in intermediate process artifacts. *It is important to note that the PRA method does not and cannot provide any assurance that the emergent property exists, e.g., that the system is safe.* It only provides indicators that there is a risk that the product will not satisfy the emergent property, i.e. that the system will not be safe. It provides insights into why and how the exposed issues might be fixed while the system is still under development. Thus this approach is meant to be used in conjunction with methods that test the final product for the emergent property.

We have applied the PRA method to three case studies of software safety processes on the NASA Constellation program, a large, network-centric Department of Defense system of systems and a small NASA satellite program. We uncovered several risks in the software safety processes of these programs. The risks in these processes shared overlapping themes: difficulty in identifying and tracking software safety concerns; inadequate traceability from safety requirements to design controls; and inconsistent scope and detail in reporting safety concerns. The feedback to the system engineers or the QA team was deemed valuable and work was underway to make the modifications presented.

As part of ongoing and future work, we will apply our process across a larger number of case studies, environments and organizations to both show that the process does find safety risks, and to understand how prevalent these risks seem to be across the industry. In addition, we are currently applying the PRA method to identify process risks in achieving other emergent system properties, such as reliability and security.

APPENDIX A – THE HAZARD ANALYSIS PROCESS

Because both of our case studies focus on the hazard analysis process, we describe some common hazard analysis concepts. While the official documentation of the Hazard Analysis Process for the DoD and NASA programs differed, the following concepts are common to both.

Hazard analysis is a top-down approach to system safety 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 on-board computer hardware failure leads to loss of mission." A 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.

Hazards analyses focuses on the identification of several important properties:

- *Causes* – The root or symptomatic reason for the occurrence of the hazardous condition;
- *Controls* – An attribute of the design or operational constraint of the hardware/or software that prevents the cause from occuring 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 1 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 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, redundant hardware). Each control has one or more verifications (e.g. test, inspection, simulation or demonstration) to ensure that the control is appropriately implemented.

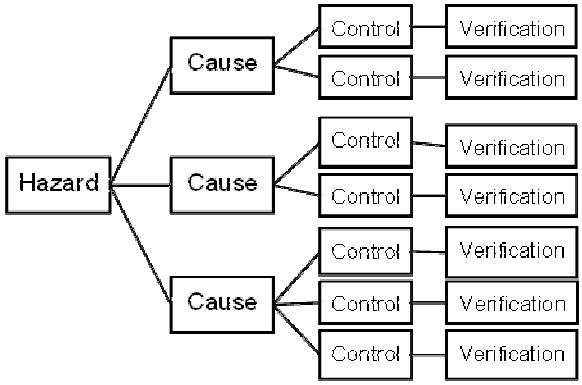

Figure 1. Hazard Structure

It is important to note that, in the DoD and NASA case studies, software is never a hazard; hazards all represent physical events that may harm the mission. Component failure (e.g., degraded thruster performance) or external 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 both the DoD and Constellation programs case studies, 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. They are also evidence of how the hazard analysis process is applied on the different projects.

Appendix B – PRA TEMPLATES

Step 1: Identify insight areas

Inputs

- The property you want to measure
- The processes associated with achieving that property
- The intermediate outputs of each step for each process

Outputs

- The set of process outputs or artifacts that should give us the most information about the effectiveness of the process for achieving the property, including:
- The format of the output
- Rationale as to how these outputs are of value for identifying the risk of non-conformance or evaluating the effectiveness of the process

Activities or Questions to ask

- What are the process outputs created during application of the process?
- What kind of information does each output provide?
- How does that information grow or change over time?
- Can I use this information to gain insight into whether the process is being performed appropriately and if the process is achieving its goals?
- Is it feasible to analyze the insight area given the current timing during the project? What is the cost of analysis?

Step 2: Identify measurement opportunities

Inputs	**Outputs**
• Process outputs/artifacts identified in step one	• Potential metrics based on process outputs/artifacts

Activities or Questions to ask

- What can I measure that will provide insight into process conformance?
- What can I measure to determine if the desired product property (e.g. safety, performance) is being achieved?
- What can I measure to evaluate if the process is sufficient for achieving the desired property?
- Can we identify potential bounds that provide insight for our goals? What is good or bad?

Step 3: Develop readiness assessment questions

Inputs	**Outputs**
• Proposed measurement opportunities and the associated risks they measure	• Advice on how the intermediate outputs and metrics can be used to identify process risk • A high-level assessment of process conformance risk, i.e. are the processes producing meaningful outputs?

Activities or Questions to ask

- Examine the process artifacts and try to apply the proposed metrics. Can I apply the metric?
    - Is the information accessible and available?
    - Is the information in good enough form that it can be measured?
    - Is the information complete?
- If I can apply a metric, then it will be a candidate for future measurement.
- If I cannot apply a metric, why not?
    - Why is the information inaccessible?
    - Why is the information in such a poor state?
    - Why is the information incomplete?

Step 4: Define goals, questions and metrics

Inputs	**Outputs**
• A set of proposed metrics that have passed the readiness assessment check	• A GQM structure with specific goals, questions and metrics

Activities or Questions to ask

- Apply the GQM method to derive a goal template, the questions, and what measures are needed.
    - What is the object of study?
    - What is the specific focus of the measure?

○ What is the purpose of the measure? ○ Who is the person who needs to make a decision about the results of this measure? ○ What are the context variables that might influence the interpretation of the results? ○ Given the goals and questions, what are the metrics?

Step 5: Develop interpretation models and define threshold values	
Inputs • A set of goals, questions and metrics to be collected	**Outputs** • A set of models that provides indication that there may be a risk
Activities or Questions to ask • Define a set of measures and interpretation models for those metrics, based upon what data is available or can be assumed, to provide indicators that there is a risk that the process is not being followed and the product is at risk of not satisfying the particular property. ○ What is the expected value of that metric and possible margin of error, i.e. what is the range of values that would be acceptable? ○ Do historical data exist for any of the metrics? ○ Are there proxies for the bounds on these metrics? ○ Can we gather any expert opinion on the bounds?	

Step 6: Propose responses to identified risks	
Inputs • Metrics and an interpretation model • Data from intermediate project artifacts	**Outputs** • Advice on what the project should do if we are outside the acceptable bounds and there is a risk
Activities or Questions to ask • Provide expert safety engineer advice on what to do under the circumstances	

APPENDIX C – DOD AND NASA SATELLITE PROJECT DESCRIPTIONS

DoD Program

The DoD system was a large, network-centric system of systems with a potentially large number of software safety risks that safety engineers needed to track and verify before the system was deployed [Basili et al. 2008]. In this system, software was critical in assuring the safety of DoD personnel[6]. The development process was expected to follow the traditional Defense Acquisition V-Model and the safety process [MIL-STD-882]. Because of the cost and complexity of developing this system, safety risk management was integrated throughout the lifecycle, since design and architecture changes late in development would be prohibitively expensive. Our goal was to develop an approach that provided the software safety engineer with early warning signs of safety problems throughout development by tracking process conformance across the multitudes of subsystem development.

We applied the PRA early in the DoD program's lifecycle, when only requirements, preliminary designs, and the safety processes were available to help gauge software safety risk. We applied *all* of the steps of the PRA *before* we had the opportunity to look at the data. As a result of this application of PRA, we modified the PRA to be an iterative process where steps 1-3 were performed (sometimes repeatedly) prior to steps 4-6.

As in the Constellation program, the DoD program development process included hazard analysis for system safety. A program-wide hazard tracking system was also in place, which was used as the basis of data for the PRA. However, in asking the readiness assessment questions, we found that the HTS contained little data. The HTS was viewed as a storage repository rather than an analysis tool by the engineers

[6] We are legally unable to provide a more informative description of the system.

– something to be used as a library after the fact rather than a tool to support safety understanding. Consequently, very little useful hazard information was available. Hazard data was unavailable, difficult to extract because of a multitude of formats, and no automated syntactic analysis of the data was available. There was insufficient information to identify a sample set *software* hazards or requirements for more detailed semantic analysis as the HTS and requirements management systems did not distinguish between software and non-software hazards and requirements. The entire collection of hazards and requirements was simply growing too large (already several hundred hazards) for us to perform our analysis given our available resources.

NASA satellite project

The NASA satellite is a joint project that partners with other international space agencies. Because satellites are unmanned, the safety risks are considerably fewer in number than on a manned mission such as Constellation. In addition, the launch of an unmanned earth satellite is a relatively mature technology that is well understood. Most of the safety risks occur during the pre-launch and launch phases, when explosions, structural collapses or the release of toxic fuels can potentially endanger the lives of workers on the ground. In this satellite, most of the software safety risks concerned the flight computer's involvement in controlling propulsion and in activating onboard instruments.

We applied the PRA method to the hazard analysis process during Phase II of development where the system design is reviewed and approved so that fabrication can proceed. These hazards differed from Constellation in that they were more mature and provided more clear descriptions of hazard controls. The satellite had 23 hazard reports, 5 of which were software-related. Each of the five software-related hazards had one software cause. We identified 12 non-software causes that contained software controls. In the five software-related hazard reports, 17% of the causes were software causes, but 57% of all causes were software-related (having a software control).

Acknowledgements

This research was supported by NASA OSMA SARP grant NNX08AZ60G to the Fraunhofer CESE. We would like to acknowledge the help of Frank Marotta at the US Army Aberdeen Test Center during the DoD case study and Karen Fisher and Risha George at NASA Goddard Space Flight Center for providing us support and access to people and artifacts of the Constellation and NASA satellite programs.

REFERENCES

BASILI, V. R., AND WEISS, D. 1984. A Methodology for Collecting Valid Software Engineering Data, *IEEE Trans. on Soft. Eng.* SE-10, 6, 728–738.

BASILI, V. R., AND SELBY, R. W. 1987. Comparing the Effectiveness of Software Testing Strategies, *IEEE Transactions on Software Engineering* 13, 12, 1278–1296.

BASILI, V. R., AND ROMBACH, H. D., 1988. The TAME Project: Towards Improvement-Oriented Software Environments, *IEEE Trans. on Soft. Eng.* 14, 6, 758–773.

BASILI, V. R., DANGLE, K., ESKER, L., MAROTTA, F., AND RUS, I., 2008. Measures and Risk Indicators for Early Insight Into Software Safety, *CrossTalk: The Journal of Defense Software Engineering*, 21, 10, 4-8.

CHRISSIS, M. B., KONRAD, M., SHRUM, S., 2003. *CMMI(R): Guidelines for Process Integration and Product Improvement (2nd Edition)*, Addison-Wesley Professional, Boston, MA.

DEPARTMENT OF DEFENSE, 2000. Standard Practice for System Safety, MIL-STD-882.

DEHLINGER, J., AND LUTZ, R., 2004. Software fault tree analysis for product lines, *8th IEEE Int'l Sym. on High Assurance Systems Engineering (HASE'04)*, Tampa, FL, 12-21.

FEDERAL AVIATION ADMINISTRATION, 2008. System Safety Handbook. http://www.faa.gov/library/manuals/aviation/risk_management/ss_handbook/.

KRISHNAN, M. S., AND KELLNER, M. I., 1999, Measuring process consistency: implications for reducing software defects, *IEEE Trans. on Soft. Eng.* 25, 6, 800-815.

LAYMAN, L., BASILI, V. R., ZELKOWITZ, M. V., AND FISHER, K. L. 2011. A Case Study of Measuring Process Risk for Early Insights into Software Safety, *Proc. of the 33rd Int'l. Conf. on Software Engineering*, Honolulu, HI, 623-632.

LEVESON, N.G., AND TURNER C. S., 1993. An investigation of the Therac-25 accidents, *IEEE Computer,* 26, 7, 18-41.

LUTZ, R. R. AND SHAW, H-Y., 1999, Applying Adaptive Safety Analysis Techniques, *Int'l. Symp. on Software Reliability Engineering*, Boca Raton FL, 42.

MAIER, T. 1995. 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.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION, 2004. NASA Software Safety Guidebook, NASA-GB-8719.13.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION, 2004. NASA Software Safety Standard, NASA-STD-8719.13B.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION, 2004. Software Assurance Standard, NASA-STD-8739.8.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION, 2007, NASA Space Flight Program and Project Management Requirements, NPR 7120.5D.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION, 2009. Constellation Program Hazard Analyses Methodology, CXP-70038, Revision B.

NUSEIBEH, B., 1997. Ariane 5: Who Dunnit?, *IEEE Software,* 14, 3, 15-16.

PARDO, C., PINO, F. J., GARCIA, F., AND PIATTINI, M., 2011. Harmonizing assurance processes and product characteristics, *IEEE Computer*, 44, 6, 94-96.